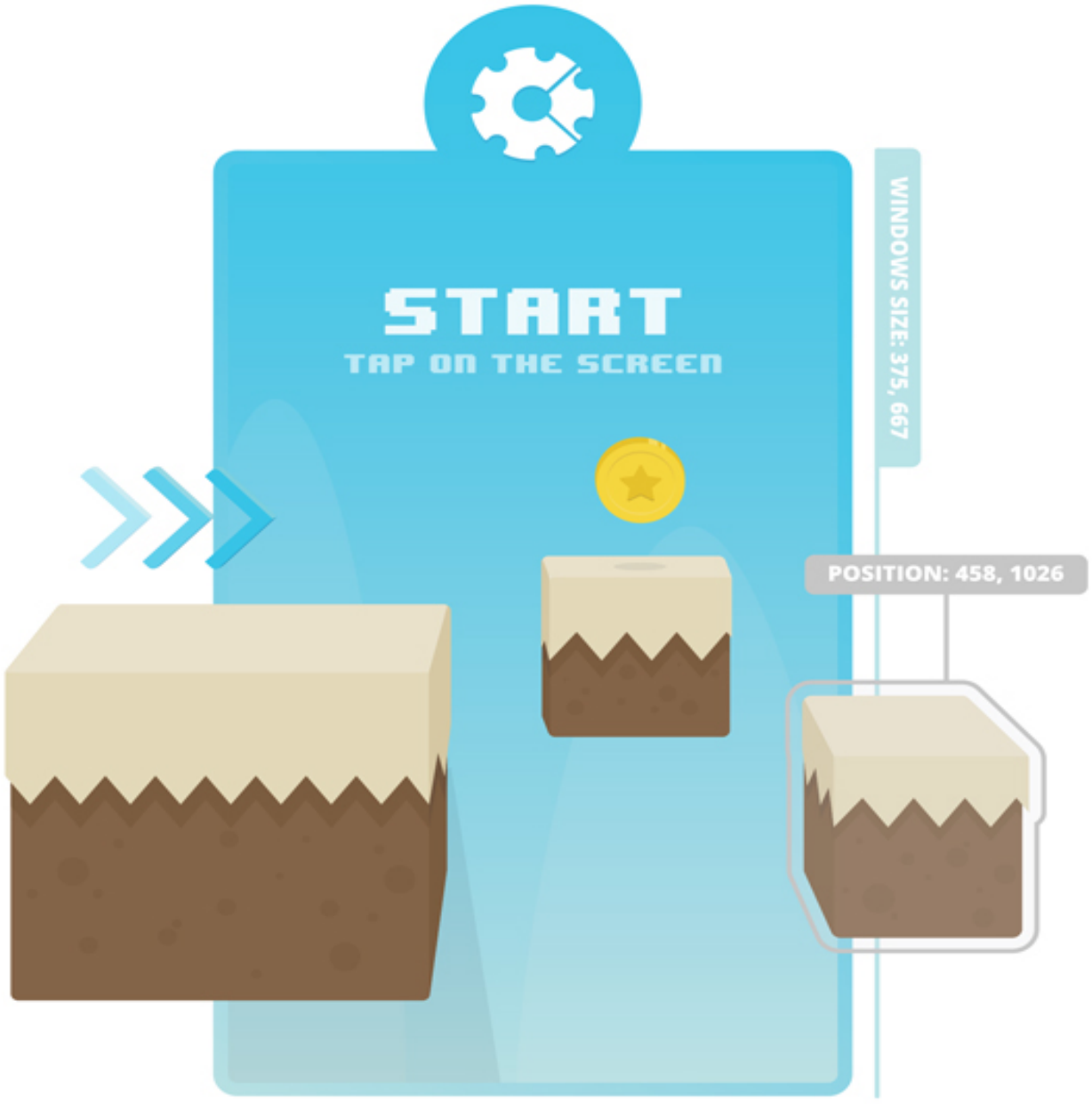
Construct 2

Crie o seu primeiro jogo multiplataforma



Casa do Código

BRUNA ESCUDELARIO
DIEGO PINHO

Sumário

- ISBN
- Dedicatória
- Agradecimentos
- Autores
- Introdução
- 1. O Construct 2
- 2. Conhecendo a plataforma
- 3. Estrutura inicial
- 4. Movimentação e animação do personagem
- 5. Sistema de vida e adição de inimigos
- 6. Telas de pausa e de fim de jogo
- 7. Inserindo itens coletáveis e pontuação
- 8. Criando efeitos e outros Layouts
- 9. Trabalhando com botões
- 10. Música e sons
- 11. Exportação e distribuição
- 12. Dicas finais
- 13. Construct 3

ISBN

Impresso e PDF: 978-85-94188-39-7

EPUB: 978-85-94188-40-3

MOBI: 978-85-94188-41-0

Você pode discutir sobre este livro no Fórum da Casa do Código: http://forum.casadocodigo.com.br/.

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

A capa deste livro apresenta o logo do Construct 2, cujo uso é oferecido por https://www.scirra.com/construct2

Dedicatória

Dedicamos este livro aos nossos pais: Maria Aparecida de Freitas Escudelario e Irineu Escudelario; Ednilda Cicilini de Pinho e Ilidio Graciano Martins de Pinho. Se não fosse por vocês, não estaríamos realizando este trabalho hoje.

Te amamos.

Agradecimentos

Primeiramente, gostaríamos de agradecer a Deus, por ter nos unido, e à nossa família, pelo constante apoio e incentivo.

Também gostaríamos de agradecer à editora Casa do Código por nos dar a oportunidade de compartilhar nosso conhecimento. Em especial, um agradecimento à Bianca Hubert, que nos auxiliou desde o início da construção deste livro.

E por fim, mas não menos importante, gostaríamos de agradecer a você que está lendo! Obrigado pela confiança! Esperamos que este livro lhe auxilie na criação dos seus primeiros jogos!

Autores

Figura -1.1: Bruna de Freitas Escudelario

Desenvolvedora Web com experiência na área de front-end, com as tecnologias e os frameworks: HTML5, CSS3, JavaScript, Node.js, Gulp.js, entre outros. Fez bacharelado em Ciência da Computação pela Pontifícia Universidade Católica de São Paulo (PUC-SP).

Sempre gostou muito de ler e estuda diariamente por meio de cursos, artigos e vídeos na internet. Começou a se aventurar no desenvolvimento de jogos há pouco tempo, mas já acumulou experiência suficiente para tocar seu primeiro negócio na internet – junto com o Diego –, a *Time to Play*, uma loja especializada em recursos para a construção de jogos.

Figura -1.2: Diego Martins de Pinho

Desenvolvedor de software desde 2013, com experiência na área de Healthcare e domínio em tecnologias Web de front-end e back-end. Dentre as principais linguagens e frameworks com que trabalha, estão: Java, JavaScript, Spring, Hibernate, Node.js e React.js. Também tem experiência em metodologias ágeis como o Scrum e o Kanban. Hoje atua como Community Manager no iMasters.

Fez bacharelado em Ciência da Computação pela Pontifícia Universidade Católica de São Paulo (PUC-SP) e possui MBA em Gerenciamento da Tecnologia da Informação pela Faculdade de Informática e Administração Paulista (FIAP). É muito criativo e sempre se interessou não somente por tecnologia, mas também por empreendedorismo, processos e gestão de negócios.

Apesar do dia a dia corrido, sempre encontra tempo para continuar a aprender e compartilhar. Tem uma grande paixão pela educação e gosta muito de ensinar. Escreve artigos sobre tecnologia na internet,

faz apresentações em eventos e é entusiasta em programação para jogos, modelagem (2D e 3D) e animação.

Capítulo 5
Sistema de vida e adição de inimigos

No capítulo anterior, fizemos um ótimo trabalho ao adicionar movimentação e animações ao nosso personagem. Passamos por conceitos importantíssimos do Construct 2, desde o editor de imagem e seus menus para animação (envolvendo o uso de quadros por segundos) até os primeiros eventos dentro das folhas de eventos.

Utilizamos os eventos de `Keyboard` (teclado) para associar as animações aos comandos dados pelo jogador. Observamos que os eventos possuem basicamente duas partes: as condições e as ações. Além disso, também usamos subeventos para conseguir descrever condições mais específicas para a animação de cada um dos estados do nosso personagem: parado, andando (direita e esquerda) e pulando.

Neste capítulo, vamos ver outro conceito importantíssimo para o desenvolvimento de jogos dentro do Construct 2, as variáveis. Para mostrar como usá-las, levaremos o nosso jogo para o próximo patamar, adicionando um sistema de vida.

Esse sistema fará com que o nosso jogo termine se o jogador ferir o nosso personagem mais de três vezes. A seguir, adicionaremos os inimigos e associaremos esse sistema ao comportamento deles.

Preparados? Então, vamos nessa.

5.1 Adicionando vidas ao personagem

Como dissemos no início do capítulo, nossa missão aqui será adicionar um sistema de vidas para o nosso personagem, dando a

possibilidade de o jogador perder o jogo. Faremos com o que ele inicie com três vidas, que serão representadas por corações na tela (não se preocupe, pois faremos todos os passos juntos).

A ideia é que cada vez que ele colidir com um inimigo, ou sair da área visível do layout, ele perca uma vida. Uma vez que ele perca todas as três, a tela de final de jogo aparecerá e este será finalizado – "finalizado" aqui significa que o jogador perderá todo o progresso e só terá a possibilidade de iniciar o jogo do início. Para fazer tudo isso, aprenderemos um conceito muito importante, o de variáveis.

Quando falamos de computadores, variáveis são espaços na memória RAM (*Random Access Memory*) da máquina física. Essa máquina pode ser um computador desktop, um notebook, um tablet ou mesmo o seu smartphone. Nesses espaços, podemos armazenar qualquer tipo de informação, desde números, palavras ou até textos inteiros.

Todo programa (ou aplicativo) que roda no seu computador utiliza dezenas, centenas ou até milhares de variáveis na memória. É por isso que, quanto maior ela for, mais coisas você consegue fazer simultaneamente.

Este espaço de memória fica fora do seu disco rígido, pois ela é uma memória de rápido acesso. Quando um programa é carregado, o sistema joga-o para dentro desta memória, e o processo usa as variáveis lá dentro para controlar o seu fluxo. E com o Construct 2, não é diferente.

Ao abrir um jogo, ele também é jogado para esta memória, e podemos utilizar variáveis para controlar o seu fluxo. Dentro do Construct, podemos declarar dois tipos de variáveis: locais e globais. Cada uma tem contexto e comportamento diferentes. Vamos ver com mais detalhes.

Variáveis locais

O primeiro tipo de variável que veremos são as locais – também chamadas de variáveis de instância. Elas possuem este nome porque estão sempre associadas ao objeto que as declararam. Mas o que isso significa?

Quer dizer que podemos ter uma mesma variável em dois objetos diferentes, com dois valores diferentes. Usamo-las para guardar informações que sejam relevantes para o objeto que as contêm, como o número de vidas deste objeto. Será que você entendeu aonde queremos chegar? Vamos ver como funcionam as variáveis locais na prática.

Começaremos adicionando uma variável local ao nosso objeto `JogadorAnimacao`. Para isso, selecione-o e vá até as suas propriedades (apresentadas na `Barra de Propriedades`). Encontre a seção `Instance variables` (variáveis de instância), como mostra a figura a seguir. Nesta opção, declaramos as variáveis que pertencerão a este objeto.

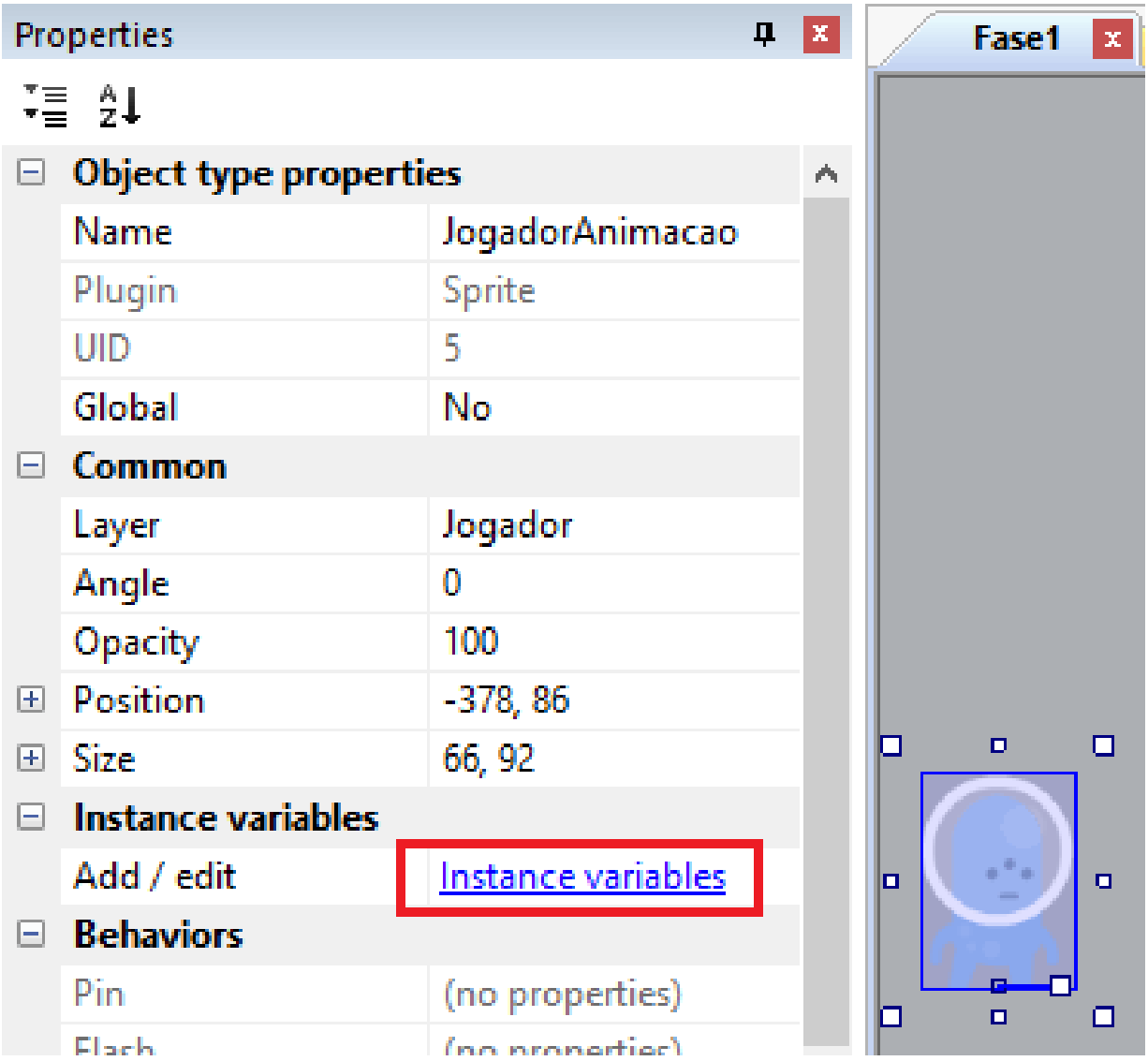


Figura 5.1: Instance variables

Na coluna `Add/edit` (adicionar/editar), clique no link onde está escrito `Instance variables`. Uma pequena janela vai se abrir oferendo algumas opções. Procure pelo símbolo de soma (`+`) e adicione uma variável nova. Na próxima janela, você encontrará quatro campos:

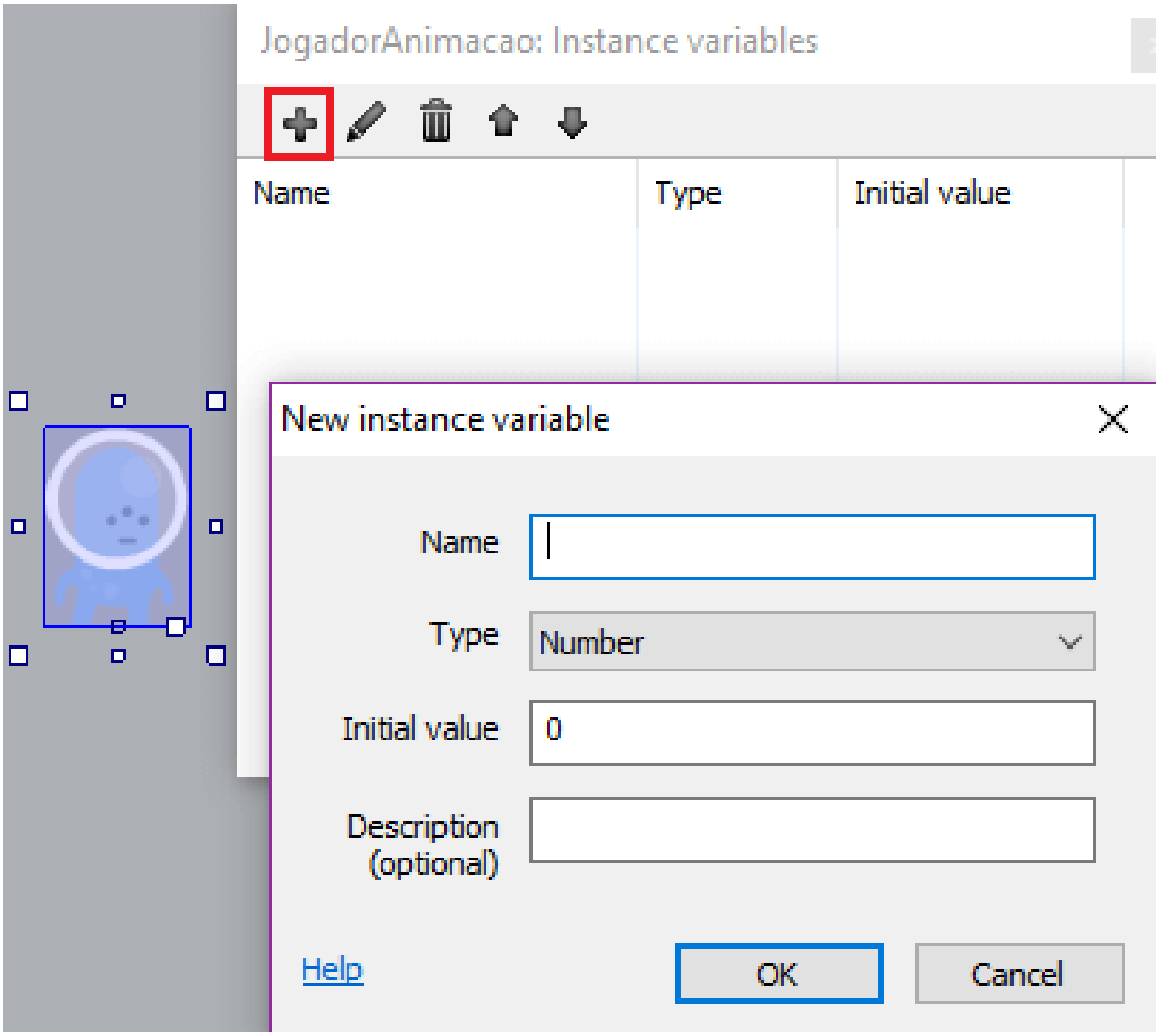


Figura 5.2: Adicionando uma variável local

- *Name* (nome);
- *Type* (tipo);
- *Initial value* (valor inicial);
- *Description* (descrição).

Como os próprios nomes dos campos já são elucidativos, não fica complicado entendermos para que servem. No campo `Name`, inserimos o nome que queremos para a variável. Para o nosso caso, coloque `vida`. No segundo campo, o tipo (`Type`), temos três opções possíveis:

- **Boolean (booleano):** este tipo só assume dois valores, verdadeiro (*true*) ou falso (*false*).
- **Number (número):** tipo específico para variáveis que representam números racionais.
- **Text (texto):** tipo que guarda palavras, textos ou mesmo um caractere.

Como estamos tratando o número de vidas do nosso personagem, escolheremos o tipo `Number`, que é o que faz mais sentido nessa situação. Mas para que você saia deste capítulo com total domínio dos tipos de variáveis que podemos criar, vamos explicar rapidamente como funcionam os outros.

O booleano representa apenas dois valores: verdadeiro (*true*) ou falso (*false*). Esse tipo de variável tem origem lá na física, em que tudo é representado desta maneira (verdadeiro é ligado, e falso é desligado). Dentro do Construct 2, podemos utilizá-la de muitas formas. Um exemplo simples seria para saber se uma alavanca foi acionada pelo jogador. Poderíamos criar uma variável chamada `acessado` com o valor `false` por padrão, e somente trocá-la quando ela fosse acionada. Desta forma, ficaria fácil saber e usar essa informação para outras coisas no jogo.

Já o tipo texto não tem segredo. Utilizamos este tipo sempre que precisamos trabalhar com textos dentro dos nossos jogos. Por exemplo, se quisermos exibir o nome do jogador dentro do jogo, podemos armazenar essa informação em uma variável do tipo texto e usá-la em qualquer lugar do jogo. Bem bacana, né?

Exploraremos um pouco mais sobre as variáveis e seus tipos no decorrer do livro, mas agora é importante que você já sabe para que elas servem. Voltando à nossa variável `vida`, escolha o tipo `Number`.

Em seguida, temos o campo `Initial value`, em que especificamos o valor inicial para a variável. Como desejamos que o jogador inicie com três vidas, qual número colocaremos neste campo? Isso mesmo, o número três.

Por fim, o último campo é totalmente opcional. Ele serve para colocar alguma descrição na variável, para que futuramente você (ou seus colegas de trabalho) não esqueça o que ela faz e para que serve.

Particularmente, não usamos muitos este campo, porque sempre preferimos criar nomes que sejam elucidativos, ou seja, que consigam descrever a sua razão, sem qualquer tipo de comentário adicional. Esta é uma prática que herdamos do mundo da computação e recomendamos fortemente, mas você está livre para trabalhar da maneira que achar melhor, ok?

Ao final deste processo, sua variável deverá ficar igual à figura:

Figura 5.3: Variável local vida

Muito bem, agora que criamos esta variável, a usaremos para saber quantas vidas o nosso personagem ainda tem. No início do jogo, ela será iniciada com três. Se levar dano de um inimigo, subtrairemos uma. Se sair da área visível, também subtrairemos uma. Ao final, se a contagem da variável for igual a zero, vamos exibir a tela de *game over* e o jogador saberá que perdeu. Bem legal, né?

Mas antes de partir para o desenvolvimento de toda essa lógica, queremos introduzir as variáveis globais. Elas são parecidas com as

que acabamos de ver, mas não estão "fixas" a um objeto, e sim ao contexto da folha de eventos em que foi criada. Ficou curioso?

Variáveis globais

Uma variável global – como o próprio nome já diz – engloba todo o escopo do jogo. O que isso significa na prática? Significa que todos os objetos e eventos podem interagir com ela, como se fossem suas próprias donas. Elas são apropriadas para guardar informações que sejam de contexto geral do jogo, como estatísticas. No nosso caso, vamos usá-la em outro caso recorrente: para guardar a pontuação do jogador.

Neste ponto, você deve estar se perguntando como saber quando usar um tipo de variável e quando usar o outro. Queríamos ter uma resposta concreta para isso, mas infelizmente não há. Tudo depende de como você quer organizar o seu jogo.

Com base em nossa experiência, utilizamos as globais somente para guardar informações que realmente sejam relevantes para o jogo como um todo, como pontuações e rankings. Por ora, siga os nossos passos e, quando se sentir confortável com a plataforma, tente outras maneiras e veja qual melhor se adapta a você.

Seguindo com o desenvolvimento do nosso jogo, vamos armazenar a pontuação do jogador em uma variável global. A pontuação do nosso jogo será feita através dos itens coletáveis que vamos inserir mais à frente. Por enquanto, vamos apenas deixar o terreno preparado.

Criaremos nossa variável na folha de eventos. Para isso, clique com o botão direito do mouse nela e selecione a opção `Add global variable` (Adicionar uma variável global). Uma janela se abrirá com o nome de `New global variable` (Nova variável global), como mostra a figura a seguir.

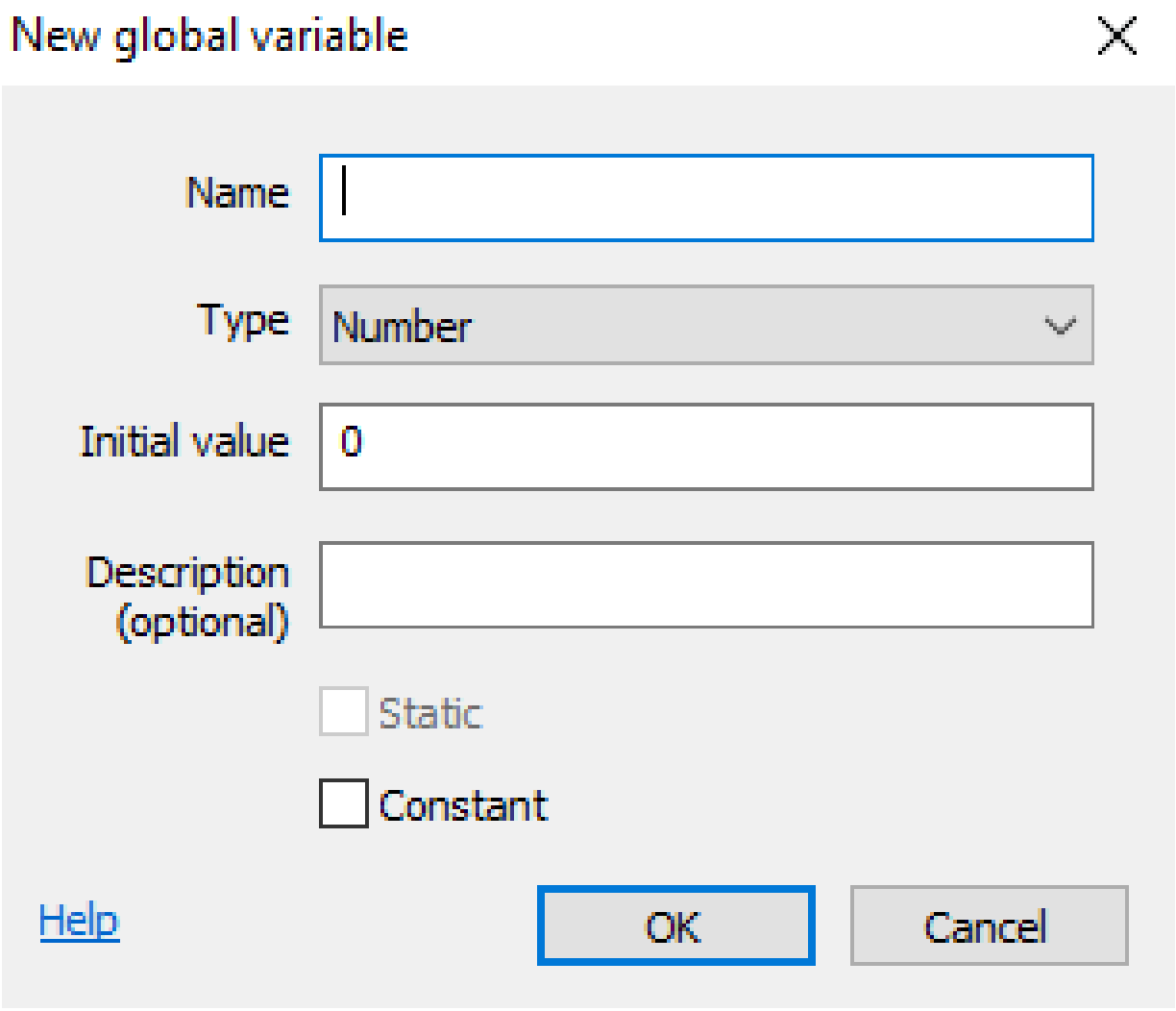


Figura 5.4: Adicionando uma variável global

Você vai reparar que ela é muito semelhante à janela que vimos na seção anterior, mas que há algumas diferenças. A primeira delas é que agora só podemos criar variáveis de dois tipos: *Number* e *Text*. A segunda é que, depois do texto de descrição, há duas caixas com as opções `Static` (estático) e `Constant` (constante).

As variáveis que são marcadas como estáticas não reiniciam os seus valores quando entram dentro do seu escopo. Seus valores são persistidos, tornando-as globais. Não se preocupe muito em entender isso agora, discutiremos isso mais à frente quando falarmos sobre grupos. Por enquanto, saiba que toda variável global

é marcada automaticamente como estática. Há alguns segredos sobre variáveis que vamos esconder de você, mas só por enquanto.

A segunda caixa torna as variáveis para somente leitura (*read only*), ou seja, não conseguimos nunca alterar seus valores, somente ler e compará-los com outros. Você pode se perguntar: "Mas em que contexto vou querer uma variável que não muda de valor?".

Sabemos que isso parece meio estranho à primeira vista, mas elas são bastante úteis, por exemplo, para gravarmos valores que são referências para outras tarefas. Poderíamos armazenar o valor da força da gravidade nela e fazer com que tudo fosse calculado de acordo com isso. Ou poderíamos armazenar o valor de uma constante para usar dentro de uma fórmula matemática, como o PI.

Há vários contextos em que elas seriam úteis. Mas fique tranquilo, não usaremos variáveis constantes. Entretanto, tenha em mente a sua utilidade.

Com a janela de criação de variáveis, preencha o campo `Name` com o valor `pontos`, o campo `type` com a opção `Number`, e coloque o número zero como `Initial value`. Fique à vontade para adicionar uma descrição e deixe as caixas a seguir desmarcadas. Ao final, ela deve ficar desta maneira:

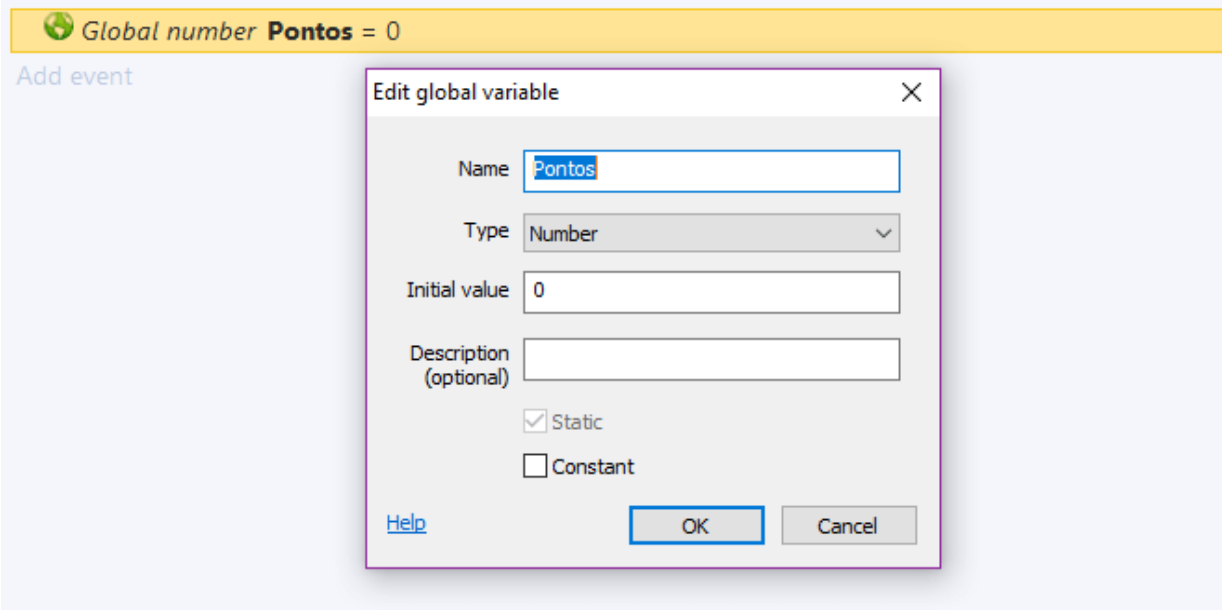


Figura 5.5: Variável global

Muito bem! Preparamos o sistema de vidas, e iniciamos o sistema de pontuação do nosso jogo utilizando as variáveis locais e globais. Nosso próximo passo será inserir os inimigos no jogo.

5.2 Inimigos

Agora que já temos o nosso personagem com movimentos, animações e vidas, vamos começar a dificultar a vida dos nossos jogadores colocando inimigos no jogo. Para este projeto, planejamos três tipos de inimigos, mas, após concluí-los, fique à vontade para deixar sua criatividade fluir e adicionar mais tipos. Por ora, siga os nossos passos, ok?

Então, ao adicionar inimigos, precisamos criar uma inteligência de movimentação e configurar cada uma de suas ações em relação ao personagem. Queremos que, quando o personagem encoste neles, perca uma vida e sinalize que recebeu dano.

Entretanto, nosso personagem não será totalmente indefeso. Também configuraremos que, se ele acertar os inimigos por cima, eles morrem. Vamos nessa?

Inserindo os inimigos

Para começar o desenvolvimento dos inimigos, adicione uma nova `Sprite` na camada `Jogador` com o nome de `inimigo1`. Agora, para ajustarmos sua animação (assim como fizemos com o personagem principal), adicione dois quadros para a animação padrão. Ao criar os quadros, importe as imagens de nome `bolinha1` e `bolinha2` que estão no diretório chamado `Inimigos`, dentro da pasta de recursos do livro.

Feito isso, nosso inimigo terá uma animação, mas ela está estranha, pois a sua velocidade padrão é muito alta para apenas dois quadros. Para ajustá-la, vá até a barra de propriedade e altere a propriedade `Speed` (velocidade). Mude o seu valor para três.

Por fim, vamos fazer com que a animação se repita para sempre enquanto o inimigo estiver vivo. Para isso, procure pela propriedade `Loop` (repetição) e coloque o valor `Yes`.

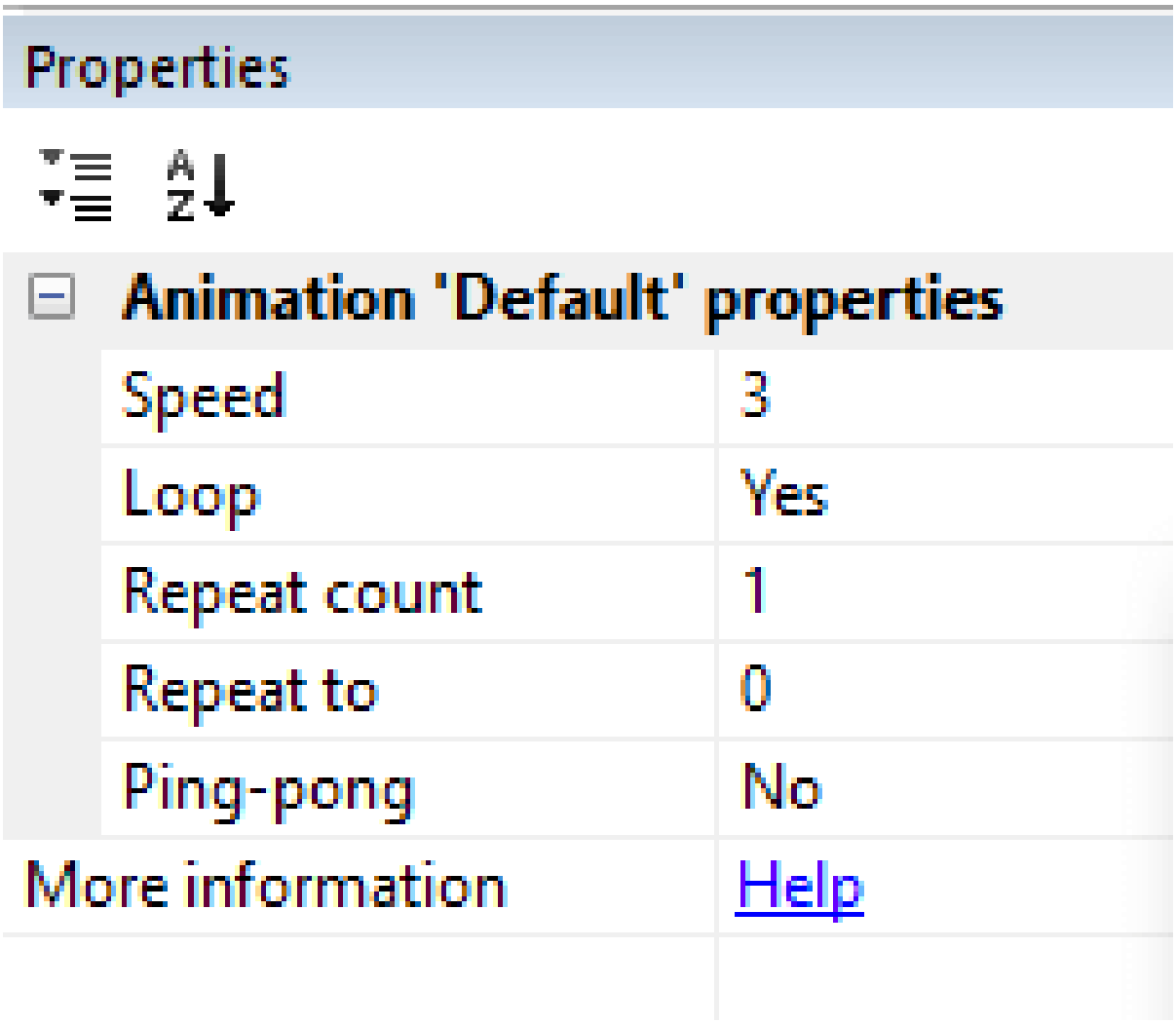


Figura 5.6: Velocidade da animação

Agora sim! Para verificar como ela está, clique com o botão direito em cima do seu nome (no caso, a `Default`), e selecione a opção `Preview`, como mostra a figura a seguir:

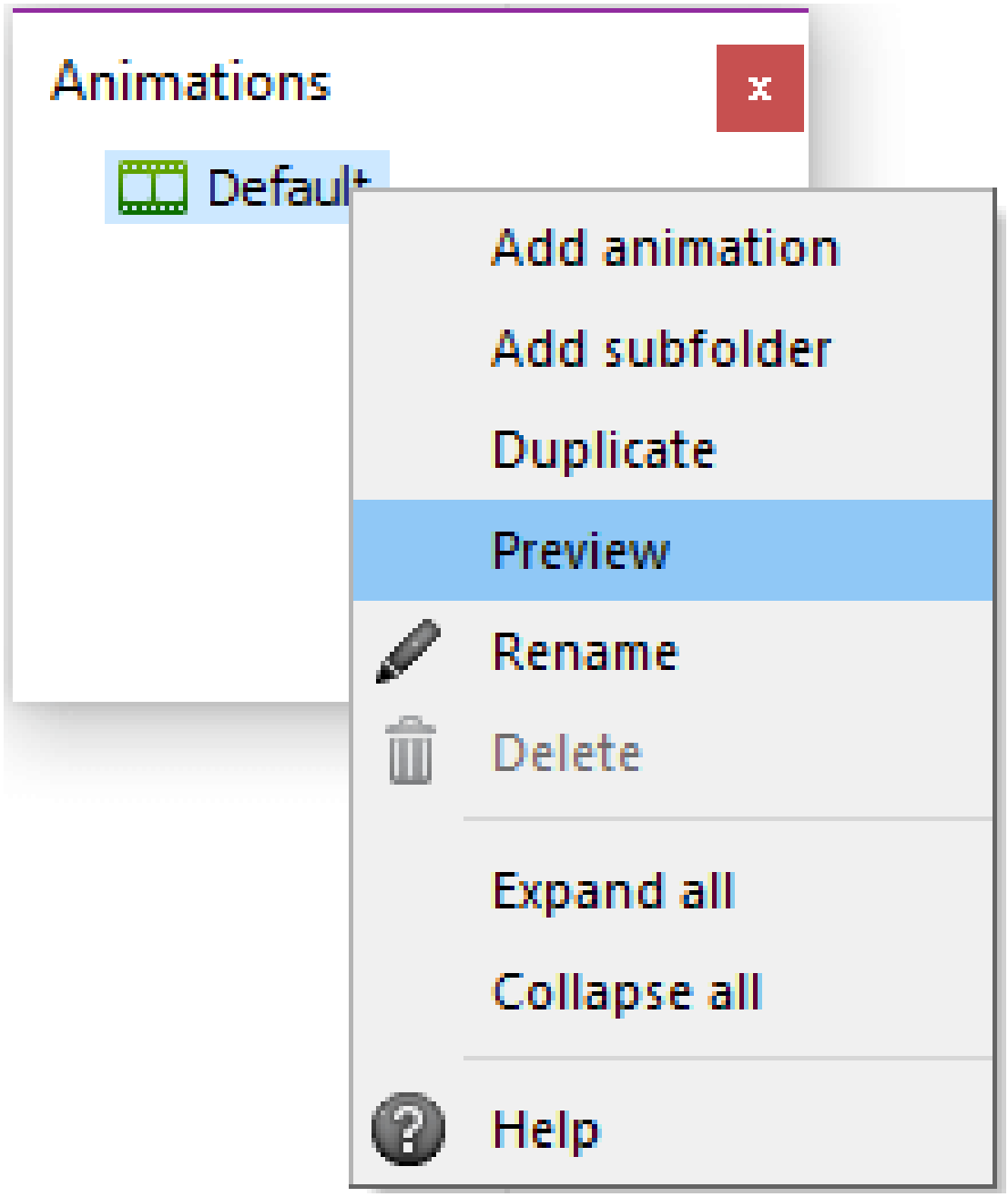


Figura 5.7: Selecionando Preview

Na pequena janela que abrir, veremos como está a movimentação do nosso primeiro inimigo do jogo.

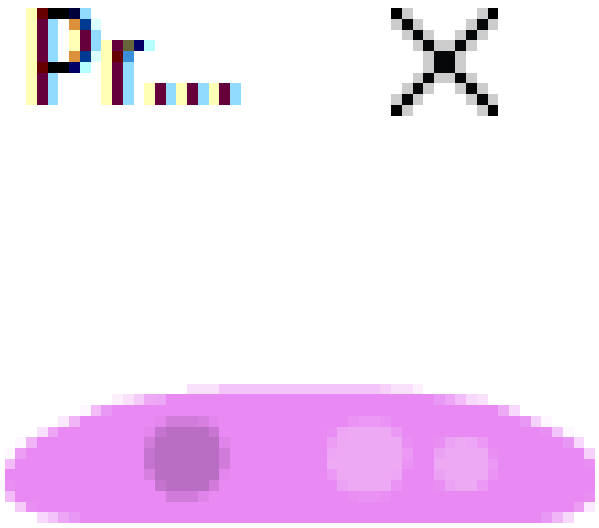


Figura 5.8: Preview da animação

Bem legal, né? Só não se esqueça de colocar o ponto de origem no centro e no meio (*bottom*) dos dois frames do `inimigo1`, como já fizemos com a maior parte dos itens que montamos até agora.

Agora repetiremos basicamente os mesmos passos para adicionar os dois outros tipos de inimigos restantes. Adicione uma outra `Sprite` para nosso outro inimigo com o nome de `inimigo2`. Novamente, insira dois quadros e coloque neles as imagens `peixe1` e `peixe2`, que também estão na pasta `Inimigos`. Configure as propriedades `Speed` e `Loop`. Por fim, adicione o ponto de origem no centro e no meio dos dois frames.

Para finalizar, faça o mesmo processo para o terceiro inimigo. Chame-o de `inimigo3` e use as imagens `abelha1` e `abelha2` nos quadros. As configurações das propriedades e o ponto de origem também devem ser os mesmos que os demais.

Figura 5.9: Os três tipos de inimigos

Muito bem! Colocamos os nossos três inimigos dentro do jogo. Mas está faltando alguma coisa, não é mesmo? Nossos inimigos ainda não têm vida nenhuma, são objetos estáticos. Na próxima seção, iniciaremos a programação de sua inteligência artificial.

Programando a inteligência dos inimigos

Já temos todos os inimigos inseridos no layout, agora podemos começar a programar sua inteligência. Você deve ter reparado, ao adicionar as imagens, que eles estão em posições diferentes. Enquanto os dois primeiros estão na horizontal, o terceiro está na vertical. Isso é proposital.

Faremos com que o segundo e o terceiro inimigos (ou seja, a abelha e o peixe) se movam na vertical, enquanto a pequena bolinha caminhe na horizontal. Ao final do processo, nossa abelha voará, o peixe pulará da água e bolinha vai rastejar pelo chão. Para alcançar isso, faremos o uso de comportamentos e variáveis.

Inicie selecionando o `inimigo1`, vá até a barra de propriedade e insira uma variável local – como já aprendemos anteriormente – do tipo `Number`. Coloque o nome `movendo` e o valor zero como padrão. Faça o mesmo para o `inimigo3`. Isso será útil para programar os eventos na próxima seção, já voltaremos a eles.

Agora selecione novamente o `inimigo1` e insira o comportamento de plataforma. Isso fará com que ele se mova pelo chão. Para o

`inimigo3`, insira o comportamento de `Sine`. Este move o objeto de acordo com o resultado de funções matemáticas, como já explicamos anteriormente. Adicione os seguintes parâmetros a seguir ao comportamento, para que ele se mova de forma vertical:

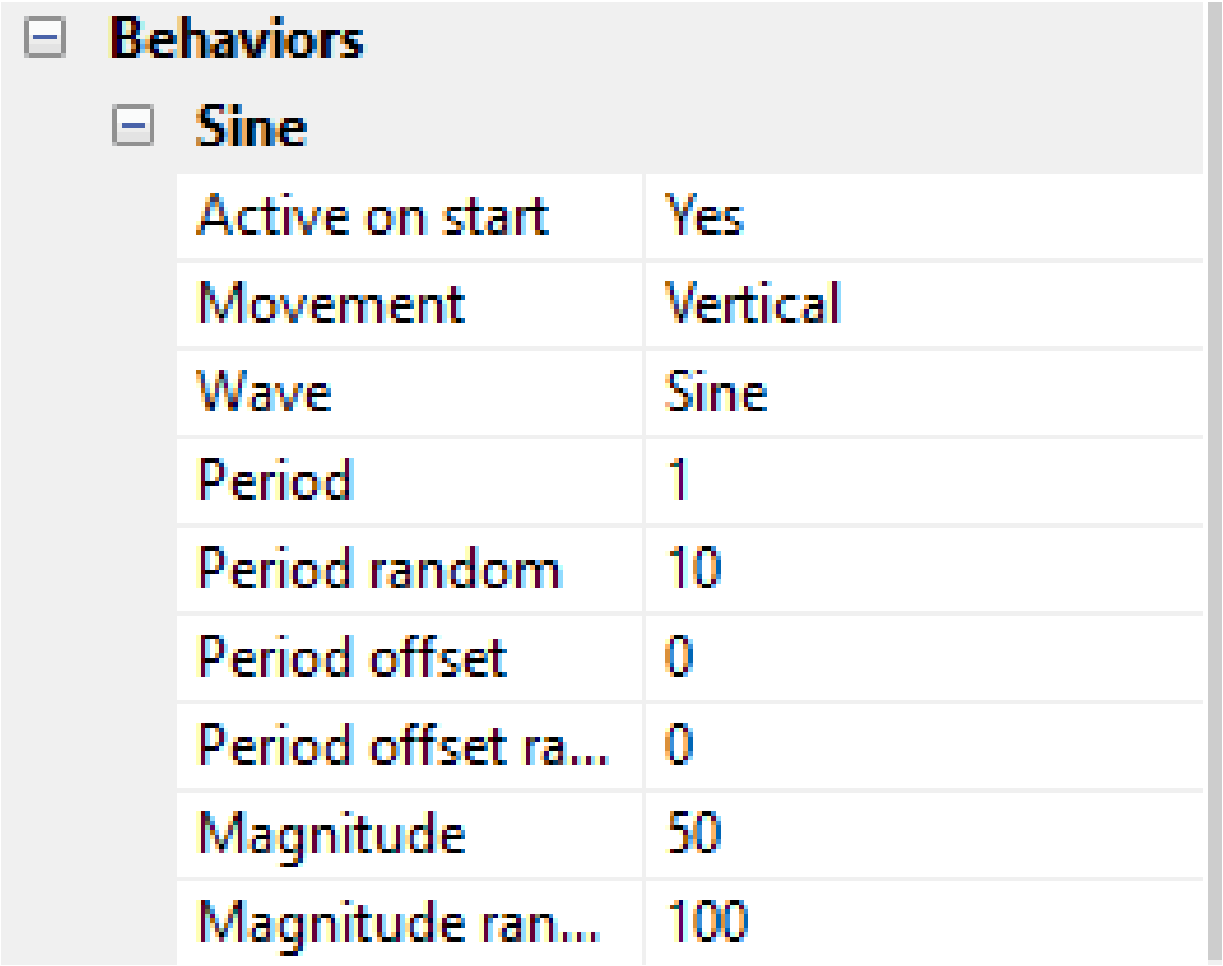


Figura 5.10: Behavior Sine no inimigo3

Maravilha, já deixamos a primeira parte da inteligência dos nossos inimigos pronta. Agora, vamos desenvolvê-los separadamente.

Programando o primeiro inimigo

Vamos adicionar os eventos referentes à movimentação do `inimigo1`. Para este, queremos que ele se mova no chão na direção

do jogador (à esquerda) assim que ele se tornar visível na tela. Ou seja, antes disso, ele deve esperar quietinho.

Para ter este comportamento, o primeiro passo é saber se ele está dentro da área visível do jogo. Faremos isso entrando na nossa folha de eventos e adicionando um novo. Selecione o `inimigo` e escolha a opção `Compare X` (comparar x). Com essa condição, conseguimos validar se a posição X do personagem está dentro da área visível.

Escolha esta opção e insira o valor `-100` (este valor não é mágico, estamos usando-o por causa do tamanho da janela que escolhemos para o jogo). Deverá ficar assim:

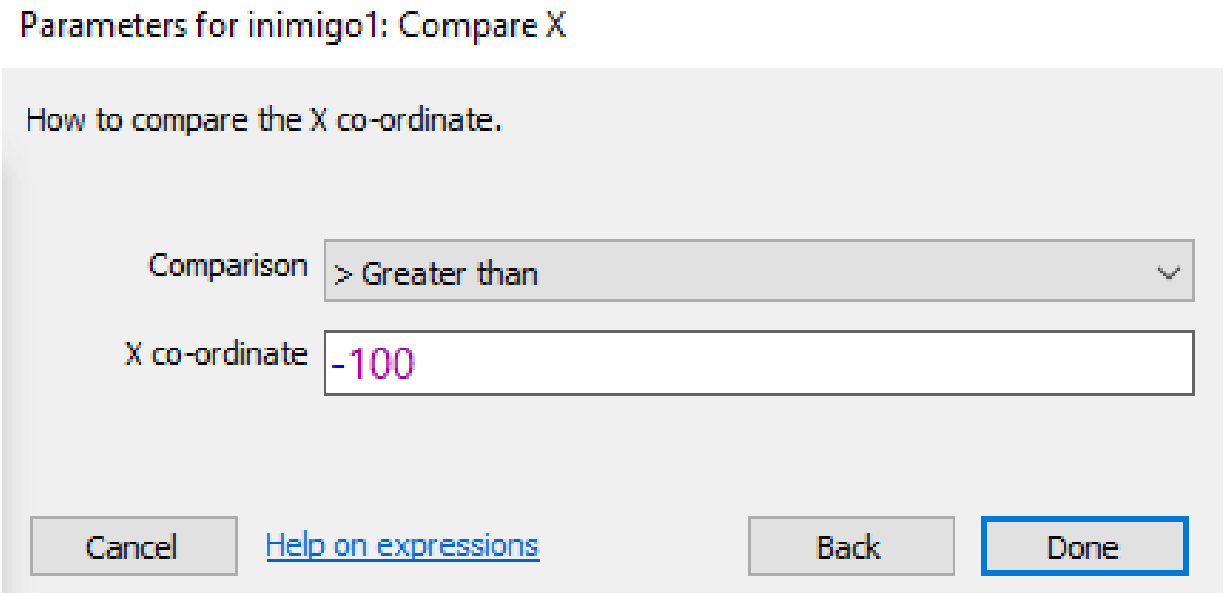


Figura 5.11: Comparando o valor de X

Colocaremos uma nova condição nesse mesmo evento, que verificará se o `inimigo1` está se movendo pela variável local `movendo` já criada. Vamos verificar da seguinte maneira: se a variável `movendo` for igual a `um` , o jogador estará se movendo; se for igual a `zero` , estará parado.

Para isso, clique no canto ao lado esquerdo e selecione a opção `Add` (adicionar) e, em seguida, `Add another condition` (adicionar nova

condição). Selecione novamente o `inimigo1` e escolha a condição `Compare instance variable` (comparar variáveis de instância). Isso fará com que possamos comparar o valor de variáveis locais.

No caso, queremos ver se a variável local `movendo` é igual a `um`. Para isso, coloque os parâmetros a seguir:

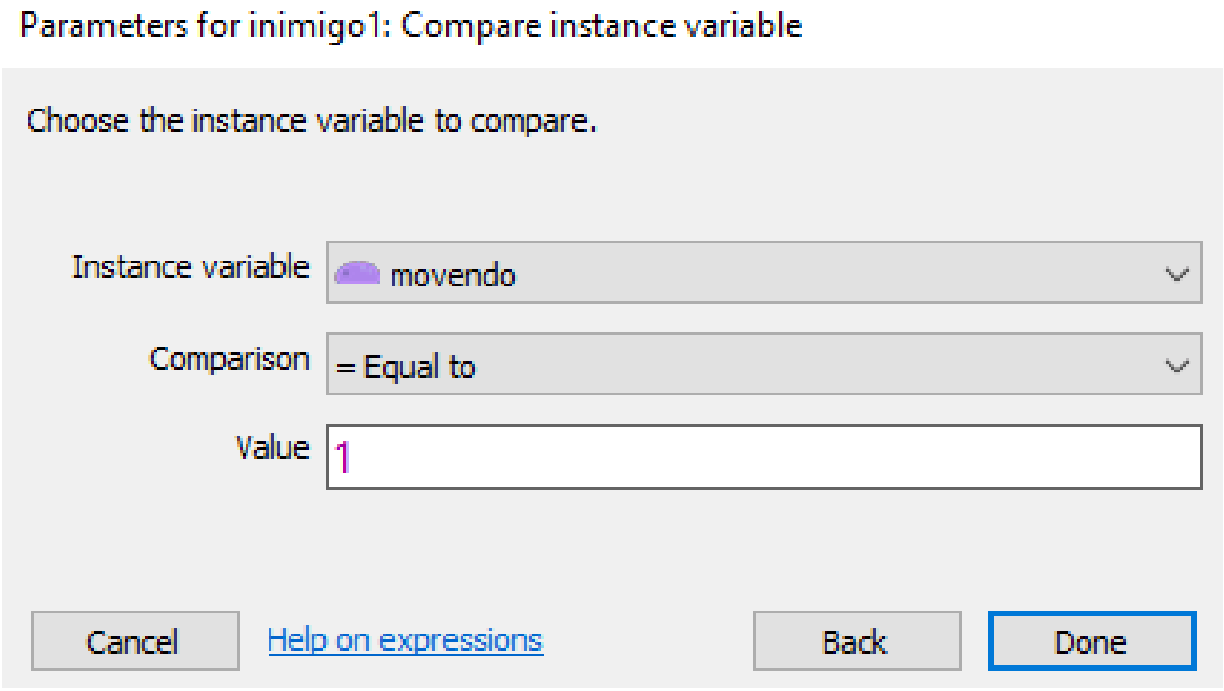


Figura 5.12: Comparando variáveis de instância

Para a ação desse evento, selecione o `inimigo1` novamente e escolha a ação `Simulate Control`. Como escolhemos o comportamento de plataforma para ele, conseguimos simular o seu movimento com essa ação. Como o nosso jogo acontece da esquerda para a direita, faremos com o inimigo se mova para a esquerda.

Para isso, na janela que abrir, coloque o valor `Left` (esquerda) em `Control` (controle):

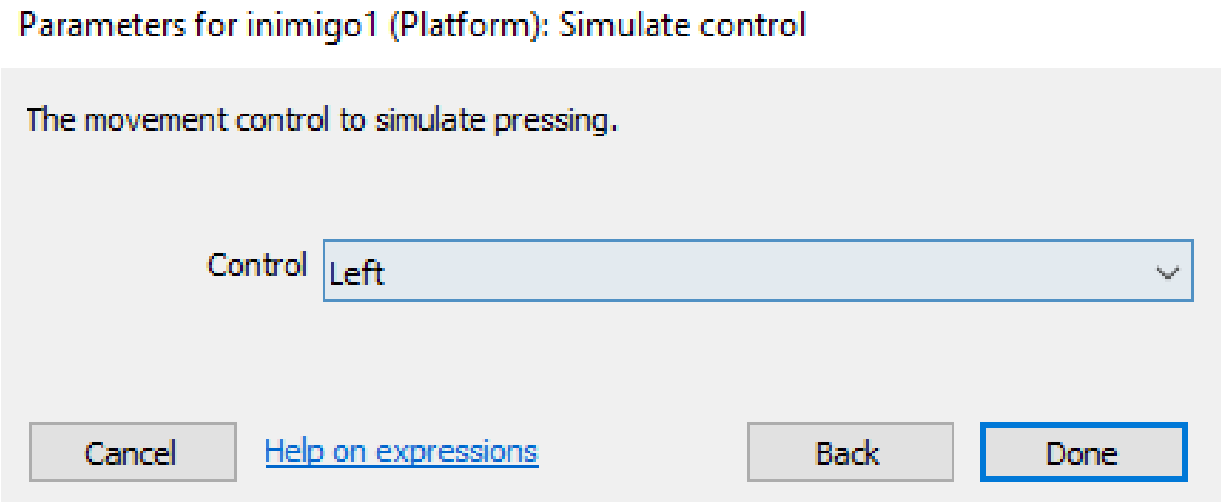


Figura 5.13: Simulate control

Então, se o `inimigo1` estiver se movendo, ele simulará o movimento de plataforma para a esquerda. Vamos adicionar outro evento para sempre setar a variável local `movendo` para `um` se o `inimigo1` estiver se movendo, como explicamos anteriormente.

Para isso, selecione novamente o `inimigo1` na janela `Add event` e adicione a condição `Is on-screen`. Para a ação desse evento, selecione o `inimigo1` e, em `Set value`, coloque o valor de sua variável local `movendo` com o valor `1`.

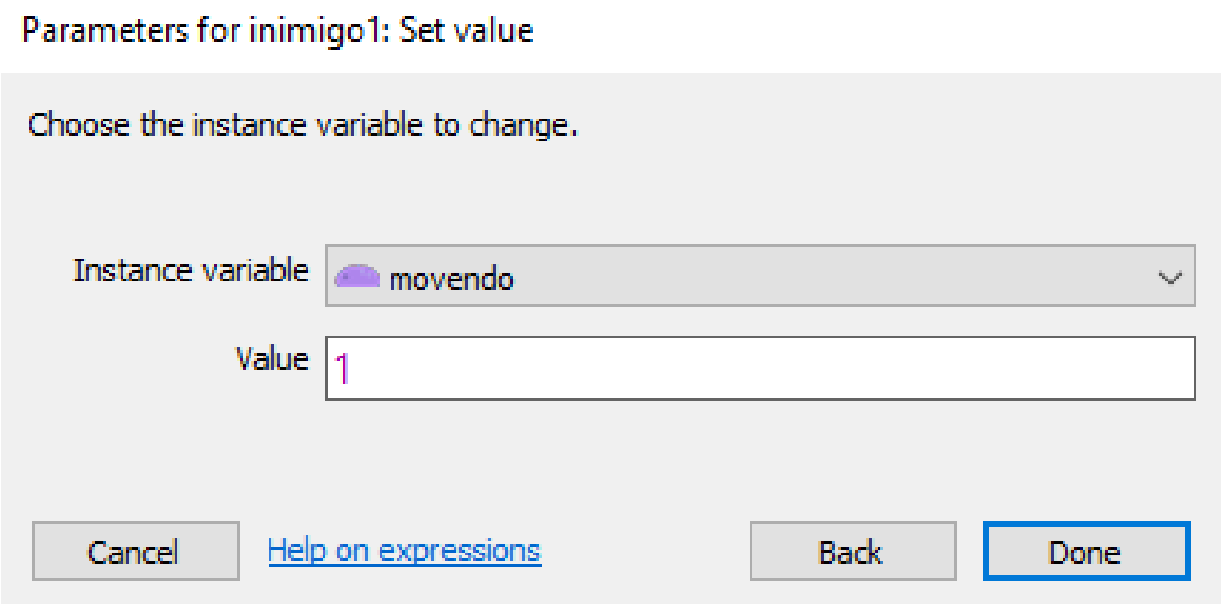


Figura 5.14: Set value

Pronto, os eventos correspondentes ao nosso `inimigo1` já estão conluídos. Ao final, sua folha de eventos deve estar assim:

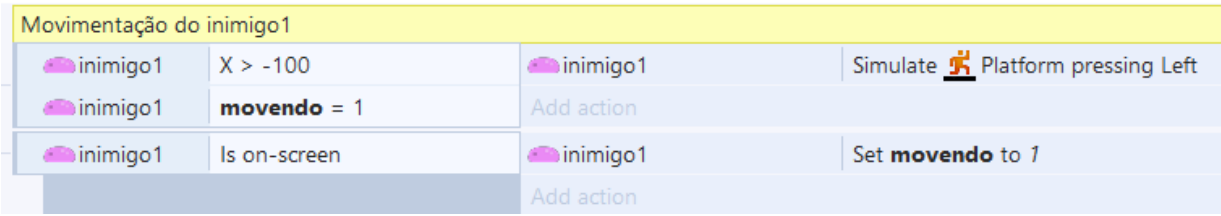


Figura 5.15: Movimentação do inimigo 1

Programando o inimigo2

Vamos partir para o `inimigo2`. Para este, usaremos três eventos diferentes.

1. O primeiro será responsável por fazer o inimigo pular a cada X segundos;
2. O segundo vai controlar o quadro de animação para quando ele estiver pulando;
3. O terceiro vai fazer o contrário do segundo.

Para realizarmos o primeiro evento, selecione o objeto `System` (objeto de sistema) na janela `Add event`. Escolha a condição chamada `Every X seconds` (a cada X segundos). Como o próprio nome diz, conseguimos definir ações para que se repitam a cada ciclo de X segundos.

Para o nosso jogo, inicialmente usaremos 1,5 segundos. Para configurar este número, coloque-o no parâmetro `Interval` (intervalo), conforme mostra a figura:

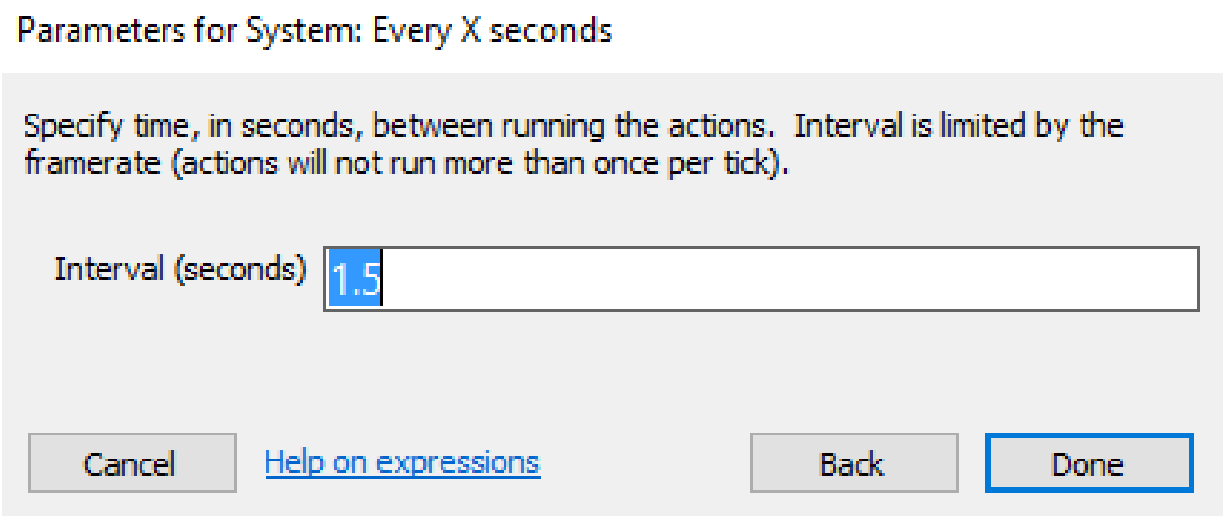

Parameters for System: Every X seconds
Specify time, in seconds, between running the actions. Interval is limited by the framerate (actions will not run more than once per tick).
Interval (seconds)
1.5
Cancel
Help on expressions
Back
Done

Figura 5.16: Every X seconds

Na ação deste evento, selecione o `inimigo2` e escolha a ação `Simulate Control`. Lembra-se de que usamos esta mesma ação no primeiro inimigo? Agora vamos configurá-la de forma diferente. Na janela que abrir, coloque o valor `Jump` (pulo) no campo `Control`. Isso fará com que o `inimigo2` pule. Muito bem, configuramos o primeiro evento da nossa lista.

Para o segundo evento, selecione novamente o `inimigo2` e selecione a condição `Is jumping` (está pulando). Ela verifica se o objeto em questão está pulando ou não. Vamos capturar esse pulo do inimigo para trocar o seu quadro de animação. Para fazer isso, na ação desta condição, selecione novamente o `inimigo2` e vá à opção `Set frame`. No campo `Frame number`, coloque o valor 0.

Lembre-se de que o zero representa o primeiro quadro da animação do personagem.

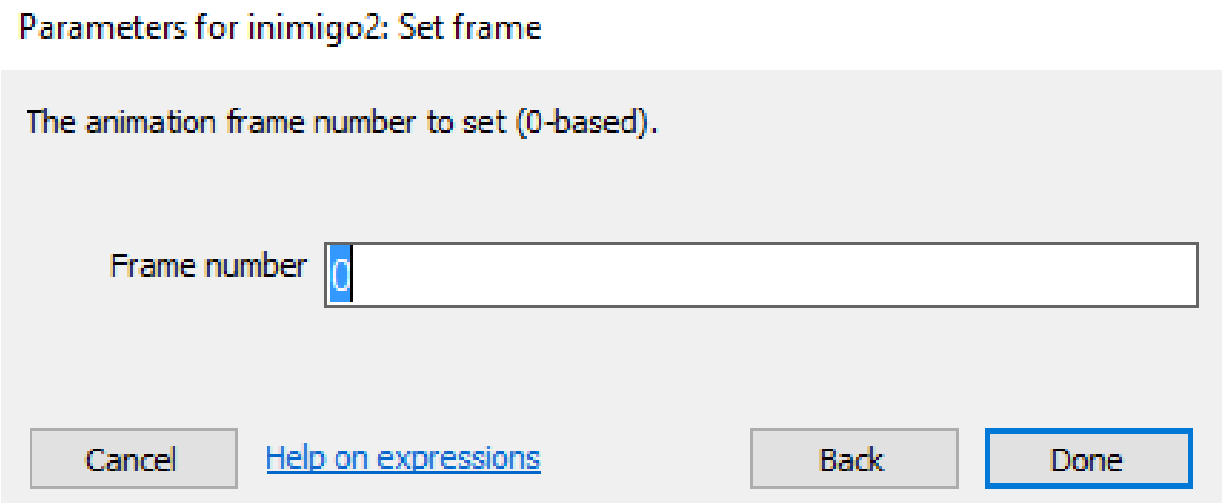


Figura 5.17: Setando o valor do frame

Feito isso, temos de partir para o terceiro item da lista, o evento contrário ao que acabamos de criar – ou seja, mudar o quadro de animação para quando o inimigo não estiver mais pulando. Selecione o objeto `System` e a condição `Else` para ele.

A condição `Else` sempre representa o caminho alternativo de uma dada condição. Como fizemos uma condição para quando o inimigo pular, conseguimos usar o `Else` para dizer o que o jogo deve fazer quando ele não estiver pulando.

Na ação selecione o `inimigo2` , vá em `Set frame` e, no campo `Frame number` , coloque o valor 1. Os eventos da movimentação do `inimigo2` devem estar assim na sua folha de eventos:

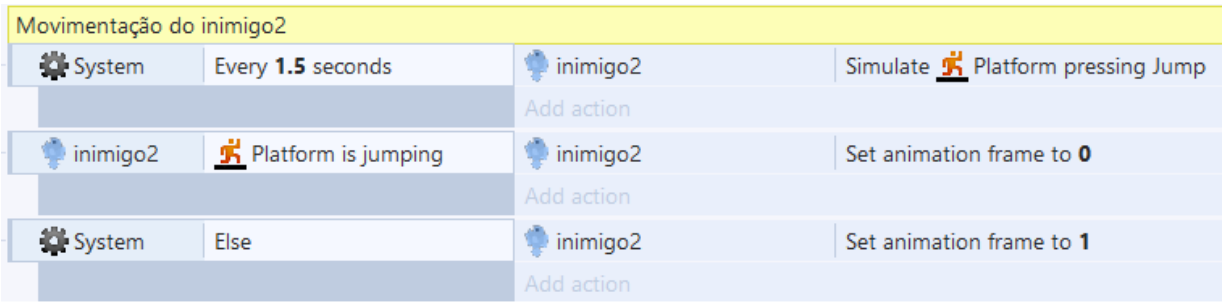


Figura 5.18: Movimentação do inimigo 2

Tudo certo? Perfeito. Vamos ao `inimigo3`.

Programando o Inimigo3

Como o `inimigo3` já tem o comportamento de `Sine` inserido, não precisamos adicionar os seus movimentos aqui. Já os configuramos nas propriedades do próprio comportamento anteriormente.

5.3 Jogador versus Inimigos

Fizemos um excelente trabalho configurando a "inteligência" dos inimigos dentro do jogo. Mas você deve ter notado que, até então, eles são inofensivos. Por que será?

Acontece que ainda não configuramos as colisões entre o nosso jogador e os seus inimigos. Ou seja, por enquanto, eles são indiferentes uns aos outros. Nesta seção, vamos alterar isto.

Jogador versus Inimigo1

O `inimigo1` será o único que o nosso personagem vai conseguir "destruir" no jogo. Faremos com que, caso o jogador consiga atingi-lo em cima de sua cabeça, ele morrerá. Este comportamento é muito comum em jogos de plataformas. Com toda a certeza, vieram

à sua mente jogos como Mario, Sonic e Crash, em que alguns inimigos são atingidos quando pulamos em suas cabeças.

Escolha o objeto `Jogador` na janela de adição de eventos (`Add event`) e escolha a opção `On collision with another object` (na colisão com outro objeto). Este evento é disparado quando um objeto entra em colisão com outro.

Vamos configurar inicialmente o que acontece quando o nosso jogador colide com o `inimigo1` . Quando isso acontecer, queremos que o nosso personagem perca uma das vidas e pisque por alguns segundos, indicando para quem está jogando que ele acabou de levar dano. Caso ele acerte a cabeça do inimigo, o jogador sai invicto.

Iniciaremos configurando o evento para quando acertarmos a cabeça do inimigo. Para isso, na janela que abrir, no campo `Object` , clique sobre `<click to choose>` , como mostra a figura:

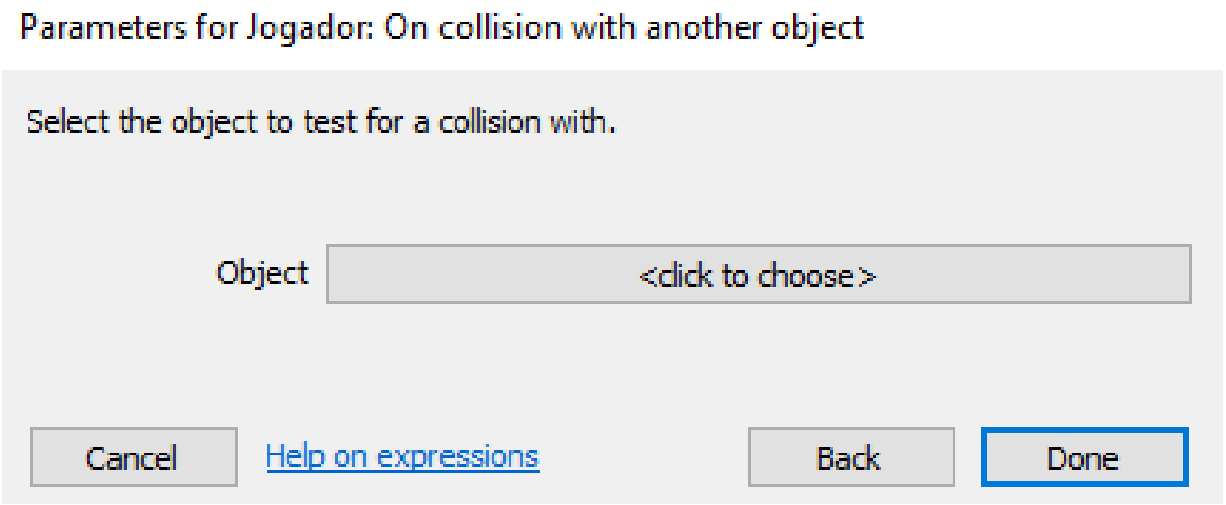


Figura 5.19: On collision with another object

Escolha o nosso `inimigo1` , como vemos a seguir:

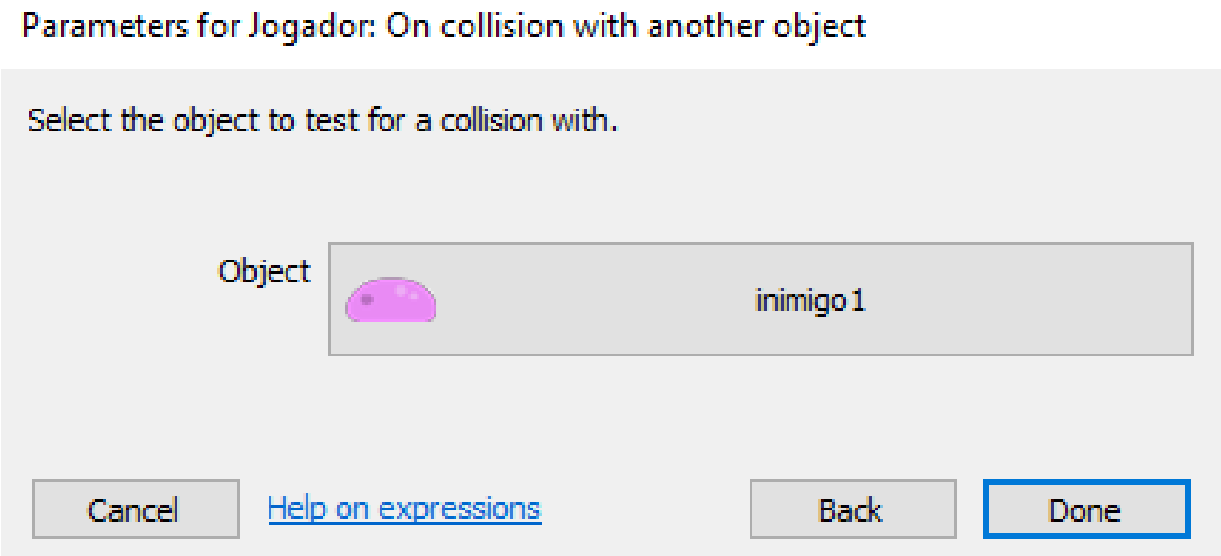


Figura 5.20: Escolhendo o objeto da colisão

Agora, vamos colocar dois subeventos nesse evento para quando o valor do Y do `Jogador` for menor que o valor Y do `inimigo1` . Isto é, quando o `Jogador` colidir por cima do `inimigo1` , este será destruído; caso contrário, se o `Jogador` estiver piscando (`Is flashing`), ele vai perder uma vida e setar o quadro de animação da `Sprite coracoes` .

Vale lembrar como criamos um subevento: clique com o botão direito do mouse no canto esquerdo da condição inserida e selecione a opção `Add` . Em seguida, clique em `Add sub-event` .

Para o primeiro subevento, queremos comparar se o valor da coordenada Y do `Jogador` é menor que a coordenada Y do `inimigo1` . Para isso, selecione o objeto `Jogador` e a condição `Compare Y` . Adicione os seguintes parâmetros para fazer a verificação do valor de Y:

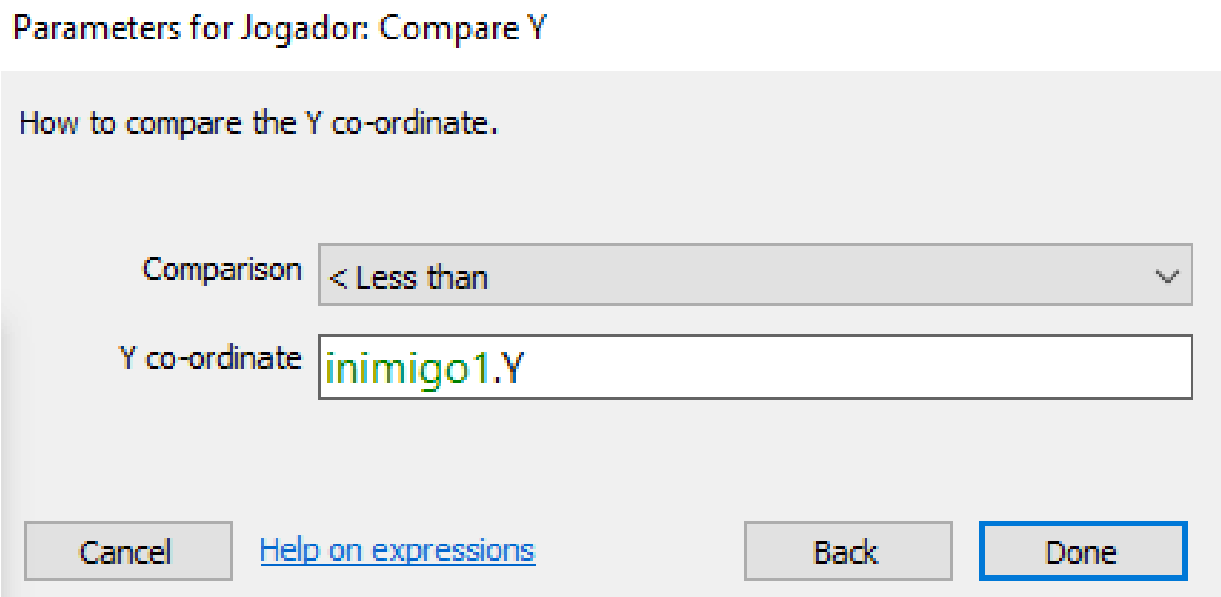


Figura 5.21: Comparando o valor de Y

Ótimo. Agora, para que a animação do personagem não fique estranha ao pular na cabeça do adversário, faremos com que ele dê um leve pulo para cima ao atingir o alvo. Conseguimos fazer isso usando a ação `Set vector Y`. Ela é responsável por configurar a posição do objeto no eixo Y.

Vamos escolher o objeto `Jogador` e a ação `Set vector Y`. Então, coloque o valor `-700` no campo `Vector Y`:

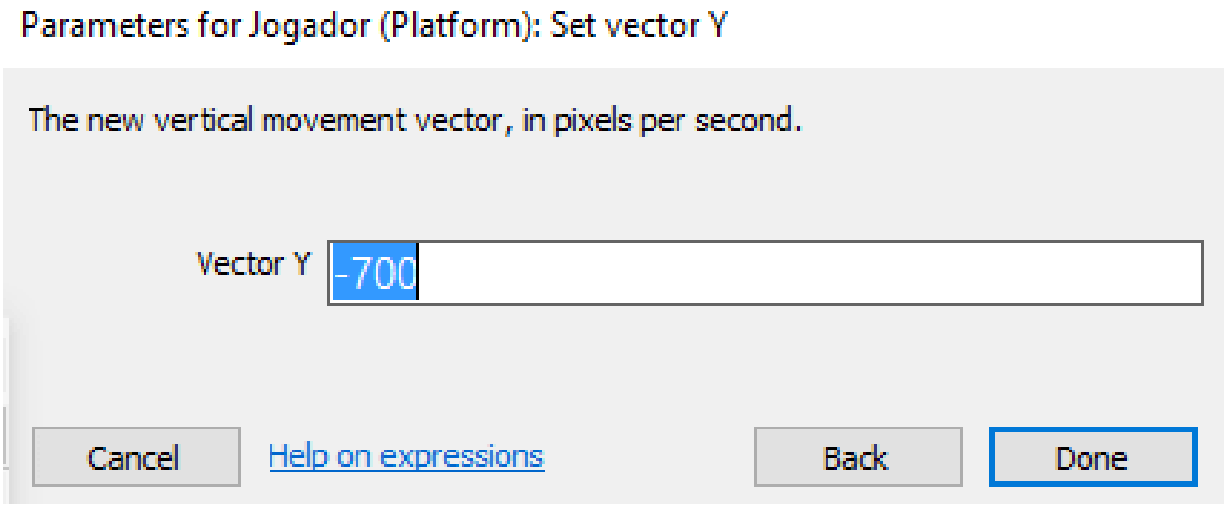


Figura 5.22: Set vector Y

Talvez você se pergunte por que estamos usando um valor negativo no eixo Y, sendo que queremos fazer com que ele pule. Fazemos isso porque o ponto de origem do plano cartesiano que define a posição dos objetos dentro do jogo está no canto superior esquerdo do seu monitor. Logo, para que o jogador vá para cima, temos de subir negativamente o eixo Y.

No nosso caso, configuramos 700 pixels. Se usarmos um valor positivo, em vez de subir, ele acabará descendo. Engraçado, né? Parece confuso a princípio, mas é só lembrar que o ponto de referência está no canto superior esquerdo do seu monitor.

Falta adicionar mais uma ação nesse evento. Nosso inimigo não pode ficar na tela depois de o destruirmos. Para removê-lo, usaremos a ação `Destroy` (destruir). Dentro dela, selecione agora o `inimigo1` e dê a ação de `Destroy`. Isso fará com o que o `inimigo1` seja destruído do jogo.

Perfeito, configuramos como eliminar o inimigo. Agora falta o segundo subevento, no qual configuraremos o que acontece quando o personagem toca o inimigo em qualquer outro lugar. Selecione o objeto de sistema (`System`) na janela `Add event`. Escolha a condição

`Else` e, no mesmo evento, coloque mais uma condição, como já fizemos anteriormente.

Clique com o botão direito do mouse no canto esquerdo da condição inserida. Selecione a opção `Add` e, depois, em `Add another condition` (ou clique em `C` do seu teclado). Selecione o `JogadorAnimacao` e a condição `Is flashing`. No evento criado, clique com o botão direito do mouse em cima do `Is flashing` e selecione a opção `Invert` (inverter); isso fará com que a ação seja invertida.

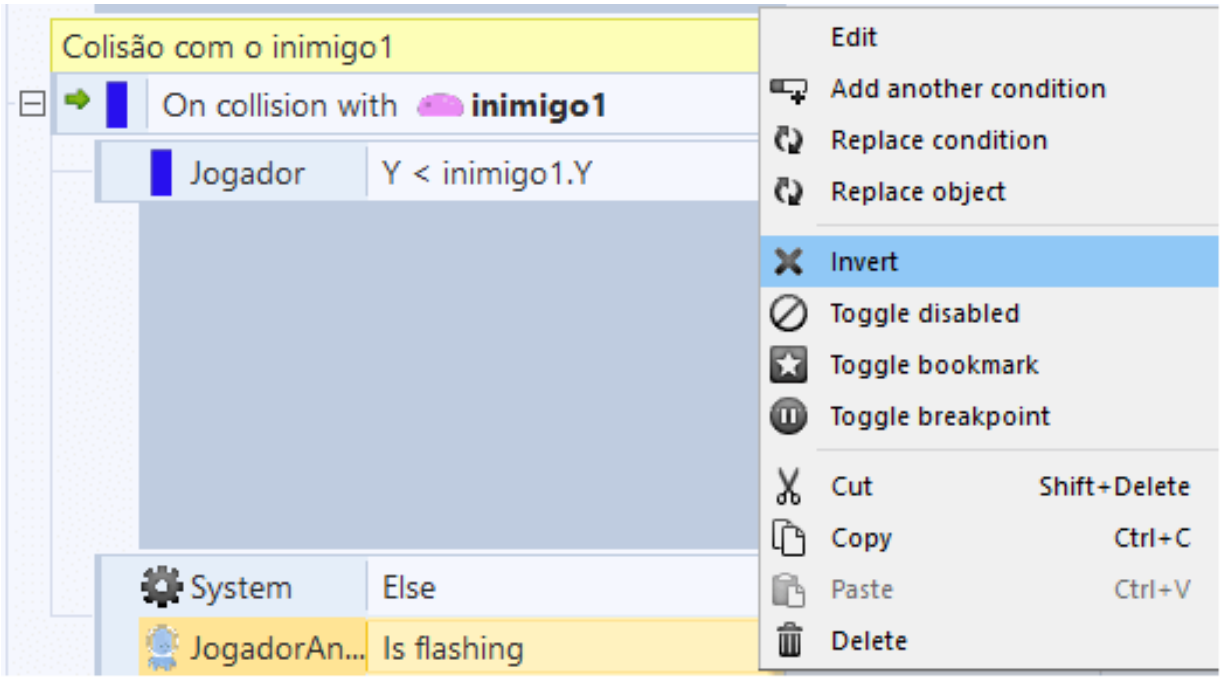


Figura 5.23: Is flashing

Só queremos que o jogador perca uma vida quando ele não estiver piscando. Para a ação desse evento, selecione `JogadorAnimacao` e dê a ação `Subtract from` (subtrair de). Isso fará com que seja subtraído um determinado valor – no nosso caso, queremos subtrair `1` da variável `vida` do nosso personagem. Para isso, coloque os parâmetros seguintes na janela que abrir:

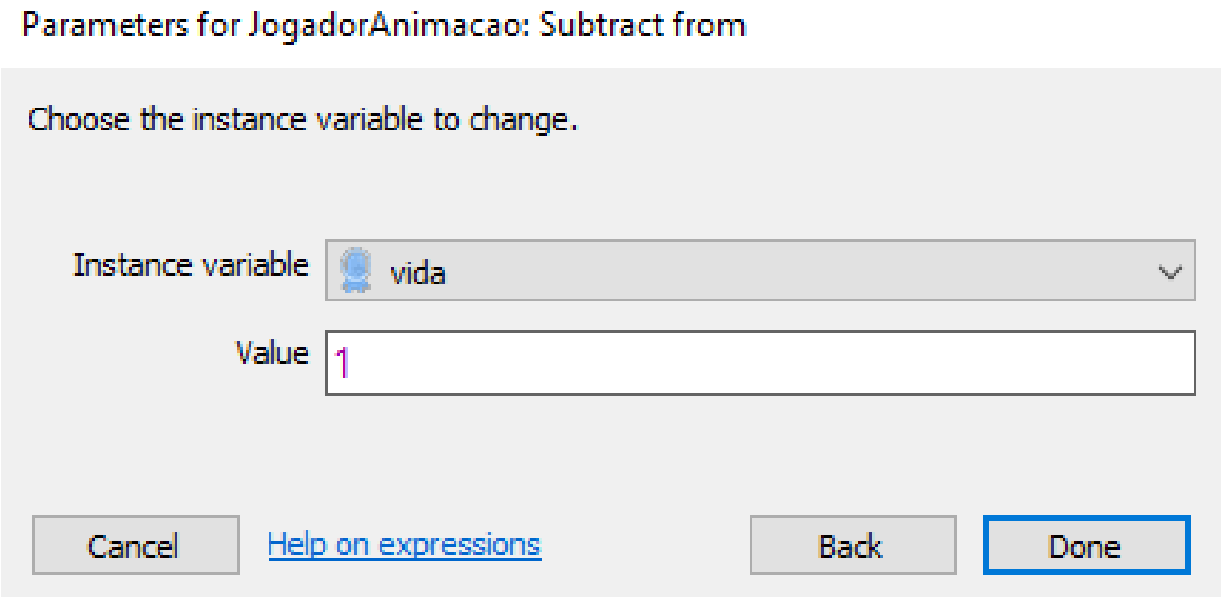


Figura 5.24: Subtract from

Na outra ação, selecione novamente o `JogadorAnimacao` e a ação `Flash`, para fazer a animação do jogador piscar. Não mude os parâmetros da janela que vai abrir em seguida, pois queremos exatamente esse valor de duração e tempo para o comportamento de piscar. Então, clique em `Done`.

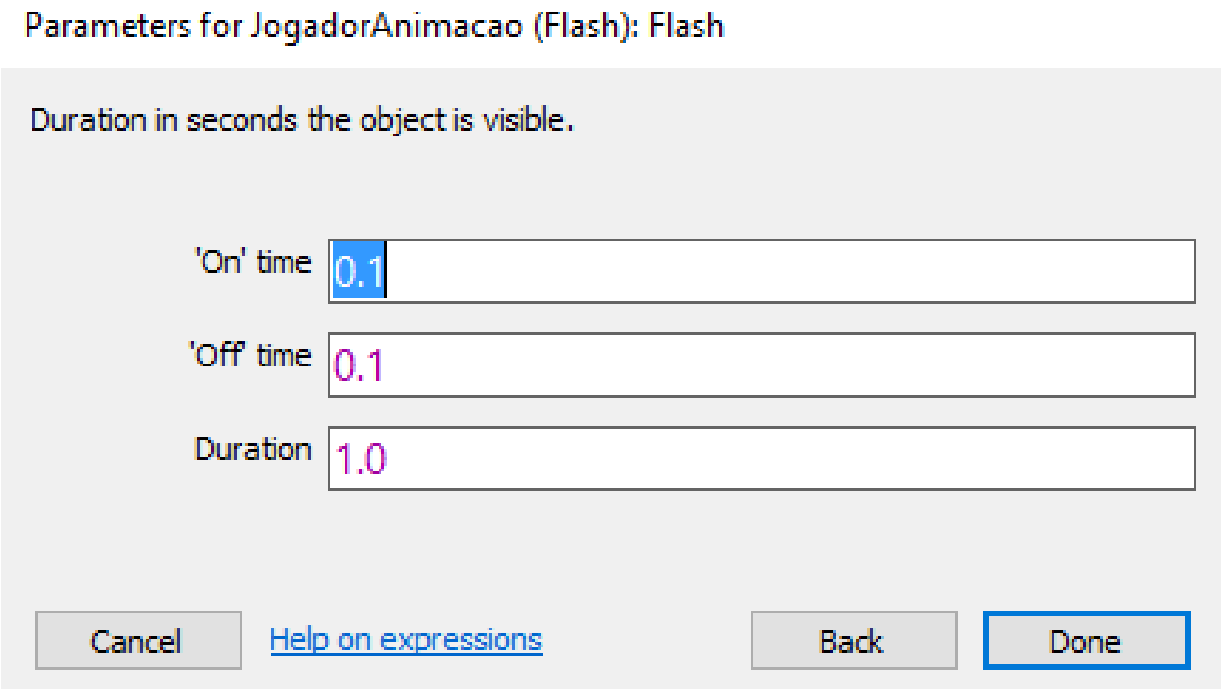


Figura 5.25: Flash

Os eventos de colisão do `inimigo1` deverão ficar igual aos da figura a seguir:

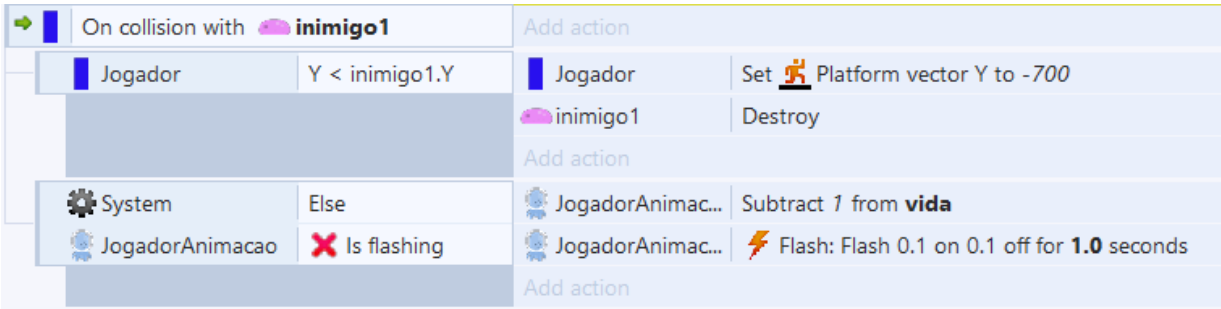


Figura 5.26: Jogador versus Inimigo1

Agora, se tocarmos no inimigo, perdemos uma vida, e o nosso jogador fica piscando por um breve período. Quando ele estiver piscando, ele fica invulnerável por um tempo, dando um espaço para o jogador se recompor. Depois deste pequeno tempo, ele fica vulnerável a danos de novo.

Para os próximos inimigos, nosso personagem não será capaz de atingi-los.

Jogador versus Inimigo2

Para o segundo inimigo, vamos criar somente um evento para, quando o `Jogador` não estiver piscando, perder uma vida ao colidir com ele. O processo é muito semelhante ao que já fizemos, então, será fácil.

Para isso, escolha o objeto `Jogador` na janela de adição de eventos (`Add event`) e escolha a opção `On collision with another object` . Na janela que abrir, escolha o `inimigo2` no campo `Object` :

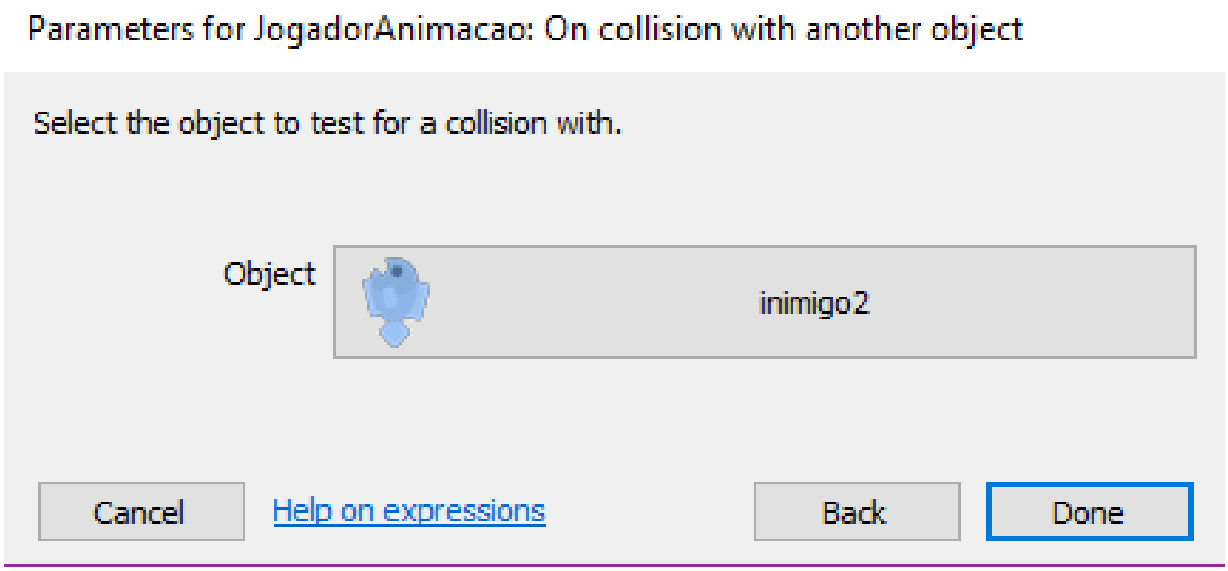


Figura 5.27: On collision with another object

Coloque uma segunda condição nesse evento e selecione o `JogadorAnimacao` e a condição `Is flashing` . No evento criado, clique com o botão direito do mouse em cima do `Is flashing` e selecione a opção `Invert` , portanto, ele só perderá vida se não estiver piscando.

Na ação desse evento, coloque as mesmas ações que colocamos no `inimigo1` para subtrair uma vida e fazer o nosso personagem "piscar". Relembre que, para subtrair um ponto de vida, selecione

`JogadorAnimacao` , dê a ação `Subtract from` e escolha `um` no campo `value` . E para fazer o jogador piscar, selecione o `JogadorAnimacao` e a ação `Flash` e, então, clique em `Done` .

Você terá o seguinte evento para o `inimigo2` :

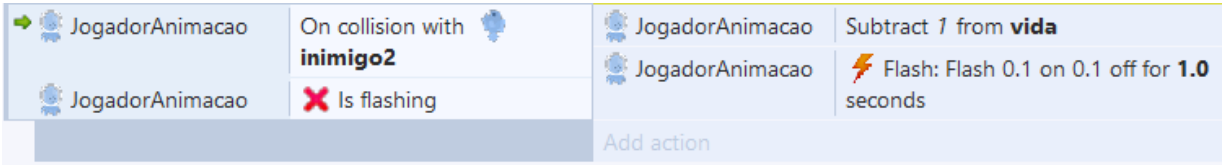


Figura 5.28: Jogador versus Inimigo2

Jogador versus Inimigo3

Para o `inimigo3` vamos fazer o mesmo evento anterior. A única diferença aqui é que colocaremos o `inimigo3` no campo `Object` , na janela `On collision with another object` , pois agora vamos programar as ações do `inimigo3`

Seu evento ficará assim:

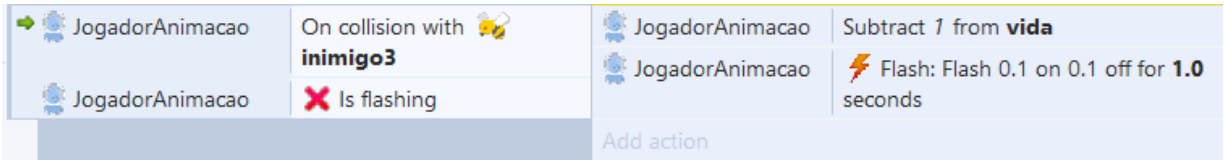


Figura 5.29: Jogador versus Inimigo3

Maravilha. Configuramos todos os comportamentos adequados para os inimigos, desde suas animações, movimentações e até o que deve acontecer quando o personagem tocar neles. Fique à vontade para alterar os valores que sugerimos e brinque com o que fizemos.

Nos próximos capítulos, trabalharemos com um sistema de pontuação e de fim de jogo, para quando o personagem ficar sem vidas.

Capítulo 1
O Construct 2

O Construct 2 é um *game engine* para desenvolvimento e criação de jogos digitais multiplataforma em 2D. Baseado em HTML5, foi desenvolvido pela empresa Scirra Ltda (https://www.scirra.com/) e lançado em 2011.

Ele é destinado a qualquer pessoa interessada na criação de jogos de forma simples, prática e com pouco conhecimento de programação e design de games. A plataforma é baseada no conceito *drag-and-drop* para desenvolver lógicas e mecanismos, e não é preciso escrever nenhuma linha de códigos em uma linguagem específica.

O usuário só precisará de um editor visual e um sistema de lógica baseada em comportamento. Também não é necessário ter conhecimentos técnicos para desenvolver, somente criatividade e vontade de tirar a ideia do papel. Por isso, esta é uma das engines mais intuitivas do mercado.

O Construct 2 consagrou-se como uma das ferramentas mais recomendadas e utilizadas por iniciantes para desenvolvimento de jogos em 2D. Essa ferramenta facilita a vida do desenvolvedor, sem abrir mão de recursos avançados, como: requisições na internet com AJAX, escrita e leitura de arquivos em XML (*eXtensible Markup Language*) e modo multiplayer (local e online). Esses e muitos outros recursos serão abordados no decorrer do livro.

1.1 Versões e licenças

Antes de continuar, é importante conhecer as versões e licenças que a ferramenta oferece. O Construct 2 encontra-se atualmente na

versão 244, e está disponível em três licenças diferentes: *Free*, *Personal* e *Business*. Vamos entender as diferenças entre cada uma delas.

Versão free (gratuita)

É possível utilizar o Construct 2 sem qualquer custo para poder conhecer, aprender e estudar. Esta versão é recomendada para quem está iniciando com a ferramenta e não pretende lucrar com as suas criações. Como é gratuita, conta com uma série de limitações, tais como:

- Impossibilidade de comercializar os jogos;
- Máximo de 100 eventos por projeto;
- Máximo de 4 camadas em um layout;
- Máximo de 2 efeitos por projeto;
- Sem organização de pastas na barra do projeto;
- Sem possibilidade de pesquisa de evento;
- Sem barra de configurações;
- Sem possibilidade de agrupar em famílias;
- Sem pré-visualização na LAN;
- Bloqueio de exportação para rodar em multiplataformas como: Android, iOS, PC ou Wii U. Só é possível exportar para HTML 5 e, posteriormente, hospedar em um domínio HTTPS (*Hyper Text Transfer Protocol Secure*).

Todos os limites da edição gratuita são removidos após adquirir uma licença comercial. No entanto, as regras para uso comercial são diferentes para indivíduos e organizações. Veja mais a seguir.

Personal License (paga)

Essa versão é recomendada para desenvolvedores independentes, ou para usuários que já possuem certa familiaridade com a plataforma e desejam comercializar suas criações. Com esta licença, os jogos podem ser vendidos até alcançar uma receita máxima de US$ 5.000,00 (cinco mil dólares) – incluindo publicidade/

promoção associada. Ultrapassando este valor, uma licença comercial deverá ser adquirida.

> É interessante saber que organizações sem fins lucrativos – como escolas, bibliotecas e universidades – podem usar essa licença para remover os limites da edição gratuita. Nessas situações, estão disponíveis descontos em massa.

Business License (paga)

Esta é a licença que oferece todos os recursos disponíveis, por isso, é indicada para empresas com alto faturamento, sem limite de receitas. O pagamento é realizado uma única vez.

Ainda sobre as licenças, vale ressaltar:

- Todas são vitalícias e associadas ao usuário, ou seja, podem ser migradas em outras máquinas.
- Todas as licenças devem ser compradas diretamente no site da Scirra ou na plataforma Steam.
- Tantos as licenças quanto os demais recursos disponíveis na loja virtual da Scirra podem ser comprados com a moeda local (R$) no cartão de crédito, débito ou boleto bancário. Neste último caso, o usuário só recebe sua chave após a confirmação de pagamento do boleto, que pode de demorar de 1 até 2 dias úteis.
- As licenças pagas também oferecem uma série de funcionalidades extras, como: plugins, pacotes de gráficos e outras ferramentas adicionais. Você encontra todas estas informações na íntegra no site da Scirra.

VOU PRECISAR INVESTIR PARA PODER FAZER MEUS JOGOS?

Não será necessário comprar uma licença para seguir o aprendizado e resolver os exercícios propostos no decorrer do livro! Mas se futuramente você tiver interesse de comercializar suas produções, será necessário, pelo menos, a *Personal License*.

1.2 Sistemas suportados

O Construct 2 está disponível apenas nas versões do sistema operacional da Microsoft. Os requisitos de sistema recomendados são:

- Sistema operacional: Windows XP/ Vista/ 7/ 8 /10
- Processador: 2 GHz dual core
- Memória: 2 GB RAM
- Gráficos: qualquer placa NVIDIA ou AMD com os drivers atualizados
- HD: 5 GB de espaço livre

Vale a pena ressaltar que a plataforma no Construct 2 só é disponível para Windows, porém, os jogos criados por ele funcionam em qualquer sistema operacional, como Linux ou Mac. Ele também é portátil, então, você poderá instalá-lo em um cartão de memória USB, por exemplo, e levá-lo aonde quiser!

NÃO POSSO USAR O CONSTRUCT NO LINUX OU NO MAC?

O Construct 2 foi construído utilizando bibliotecas únicas no sistema operacional da Microsoft, entretanto, há alternativas para se poder trabalhar com ele em outros sistemas. A primeira opção é instalar uma máquina virtual usando programas como o VirtualBox ou o VMware.

A segunda opção é trabalhar com a versão beta do Construct 3. Ela é muito semelhante ao Construct 2, mas está em fase teste, possui um visual diferente e funciona diretamente no navegador. Falaremos melhor sobre isso mais à frente.

1.3 Instalação

Há duas maneiras de instalar o Construct 2 em sua máquina: pelo download do executável no site oficial da Scirra, ou por meio da plataforma Steam.

Download pelo site da Scirra

Se optar por fazer o download no site da Scirra, primeiramente acesse https://www.scirra.com/construct2 e clique no botão `Free Download`, conforme mostra a figura:

Figura 1.1: Download no site da Scirra

Quando for redirecionado, o download deve ser iniciado automaticamente. Caso isto não ocorra, basta clicar no `Try again`.

Thanks for Choosing Construct 2!
You're now downloading Construct 2 r239
Download hasn't started? Try again
Whilst you're downloading...

Figura 1.2: Obrigado por escolher o Construct 2!

Após o término do download, o instalador deve ser executado e a opção `I Accept the agreement` marcada. Depois, clique no botão `Next`.

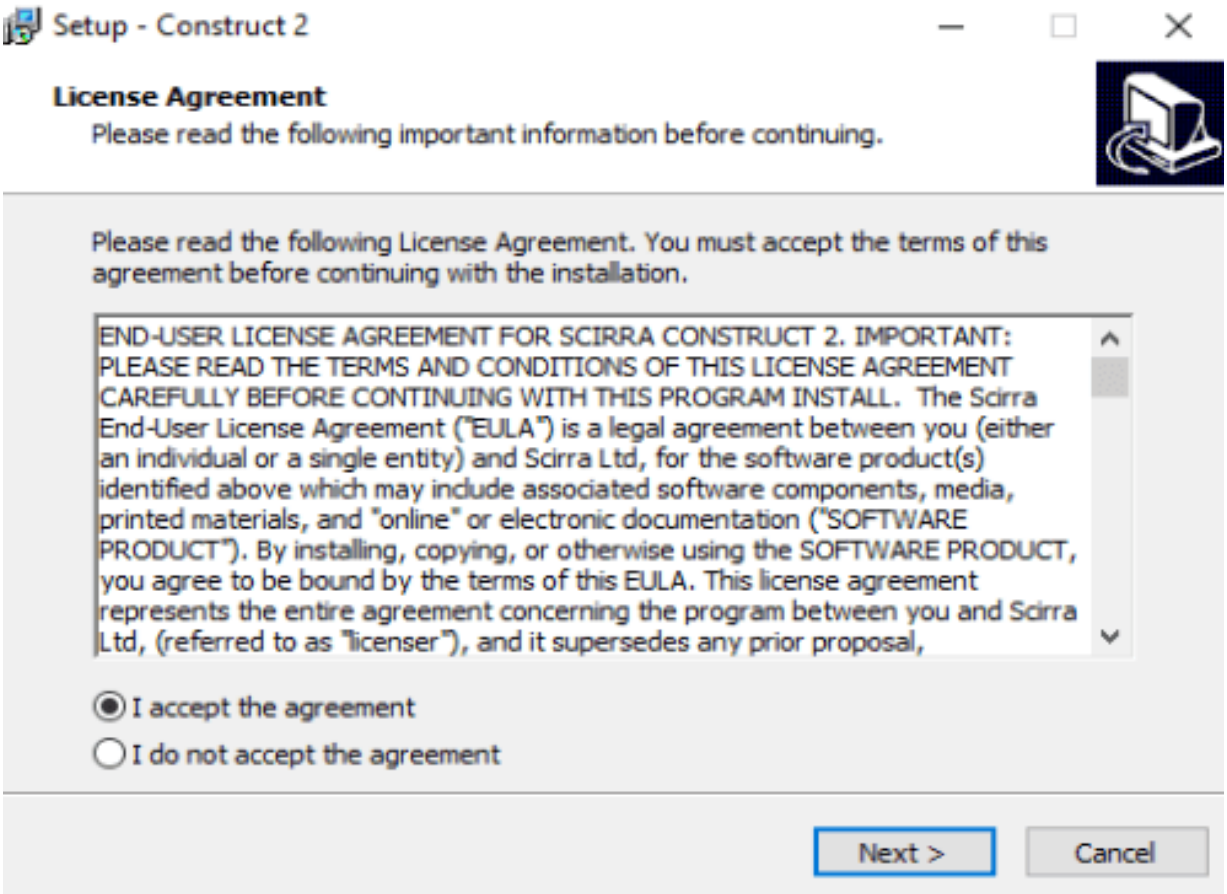

Setup - Construct 2
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
END-USER LICENSE AGREEMENT FOR SCIRRA CONSTRUCT 2. IMPORTANT: PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE CONTINUING WITH THIS PROGRAM INSTALL. The Scirra End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Scirra Ltd, for the software product(s) identified above which may include associated software components, media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. This license agreement represents the entire agreement concerning the program between you and Scirra Ltd, (referred to as "licenser"), and it supersedes any prior proposal,
I accept the agreement
I do not accept the agreement
Next >
Cancel

Figura 1.3: Acordo de licença

Agora é preciso informar qual versão será instalada, **32-bit** ou **64-bit**. Caso você não saiba qual é a arquitetura do computador, deixe selecionada a opção `Auto-detect what kind of computer I have`, que identificará automaticamente qual é. Então, clique em `Next`.

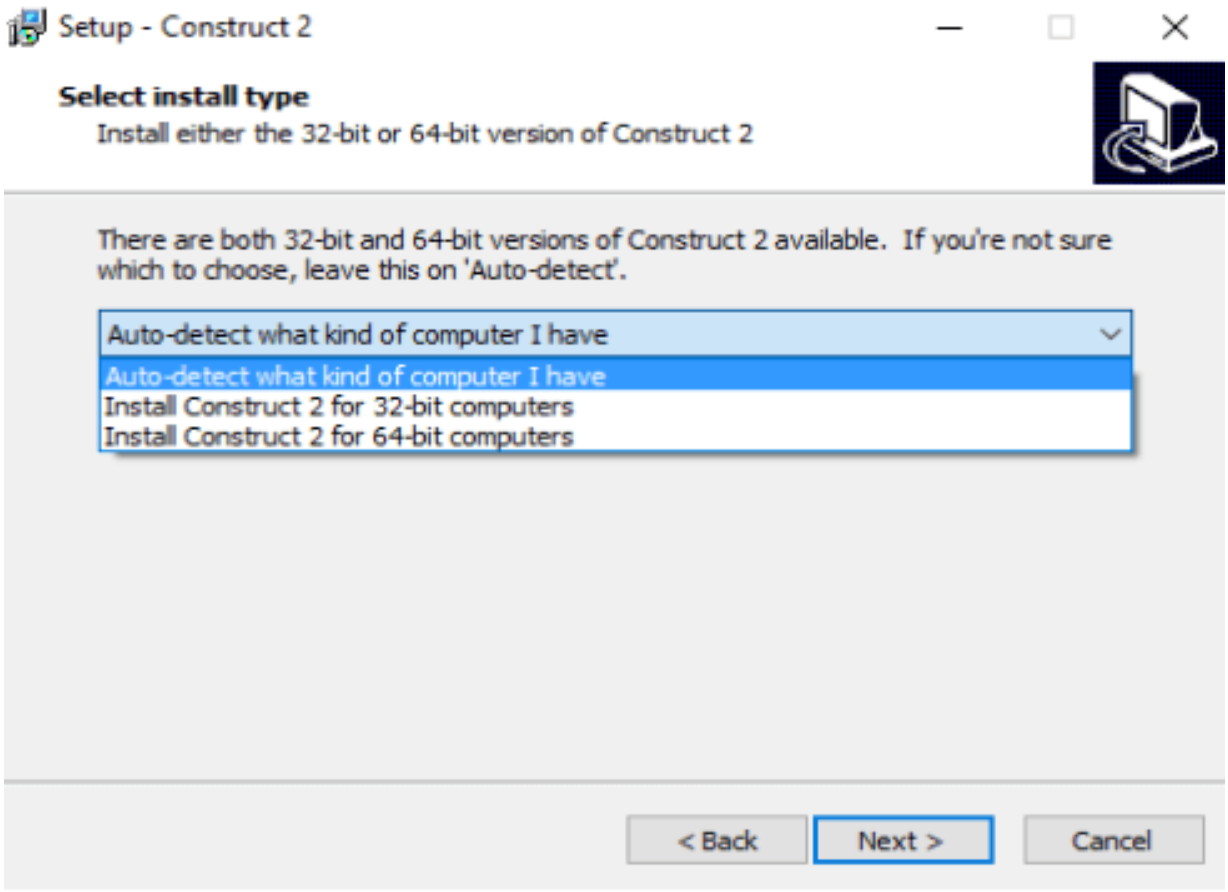

Setup - Construct 2
Select install type
Install either the 32-bit or 64-bit version of Construct 2
There are both 32-bit and 64-bit versions of Construct 2 available. If you're not sure which to choose, leave this on 'Auto-detect'.
Auto-detect what kind of computer I have
Auto-detect what kind of computer I have
Install Construct 2 for 32-bit computers
Install Construct 2 for 64-bit computers
< Back
Next >
Cancel

Figura 1.4: Tipo de instalação

Se você desejar criar um atalho na sua Área de Trabalho, marque na janela seguinte a opção `Create a desktop icon`. Em seguida, clique em `Next` para prosseguir e, por último, em `Install` para finalizar a instalação. Na última tela do instalador há várias opções, para uso de navegadores como o Google Chrome e o Mozilla Firefox recomendados pela Scirra. Após escolher a opção, clique em `Finish`.

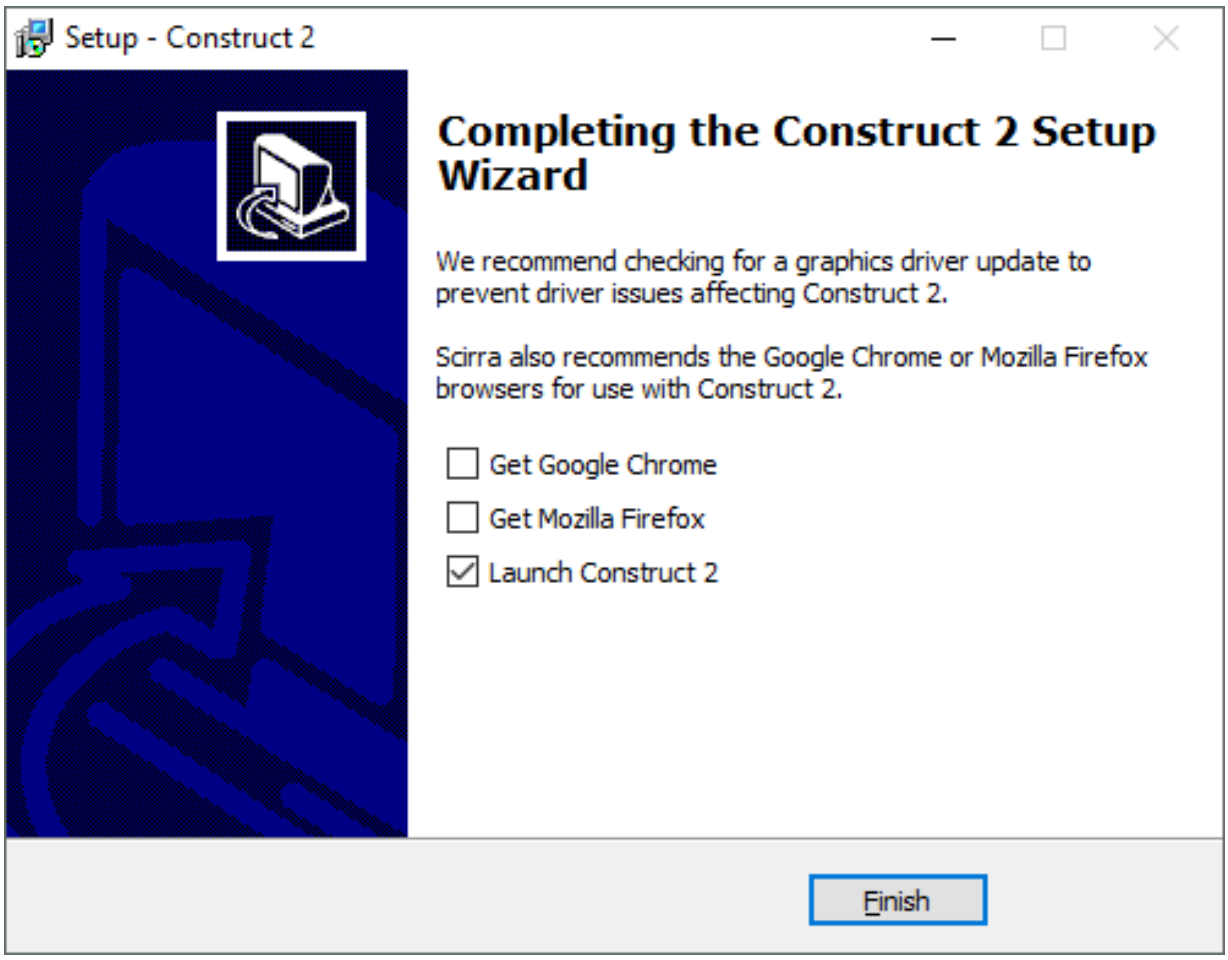

Setup - Construct 2
Completing the Construct 2 Setup Wizard
We recommend checking for a graphics driver update to prevent driver issues affecting Construct 2.
Scirra also recommends the Google Chrome or Mozilla Firefox browsers for use with Construct 2.
Get Google Chrome
Get Mozilla Firefox
Launch Construct 2
Finish

Figura 1.5: Instalação completa

Agora, o Construct 2 já está instalado no seu sistema e pronto para ser usado!

Download pela plataforma Steam

Se o download for pela Steam, primeiramente é necessário criar uma conta gratuita e ter essa plataforma em seu computador. Entre no site da Steam, em http://store.steampowered.com/, e clique no ícone superior `Instale o Steam` para baixá-la, conforme mostra a figura seguinte.

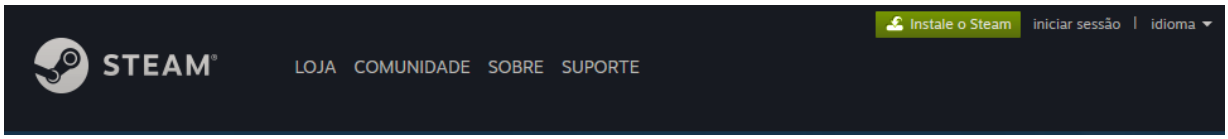


Figura 1.6: Instalação da Steam

Depois de instalada, acesse-a com a sua conta. Com a plataforma aberta, procure por `Construct 2` na loja.

Figura 1.7: Loja Steam

A seguir, abrirá uma página com informações do Construct 2 e com as opções de download, tanto da versão gratuita como das pagas:

Figura 1.8: Escolha uma licença

Depois de escolher a licença, vai abrir uma aba com as informações do download.

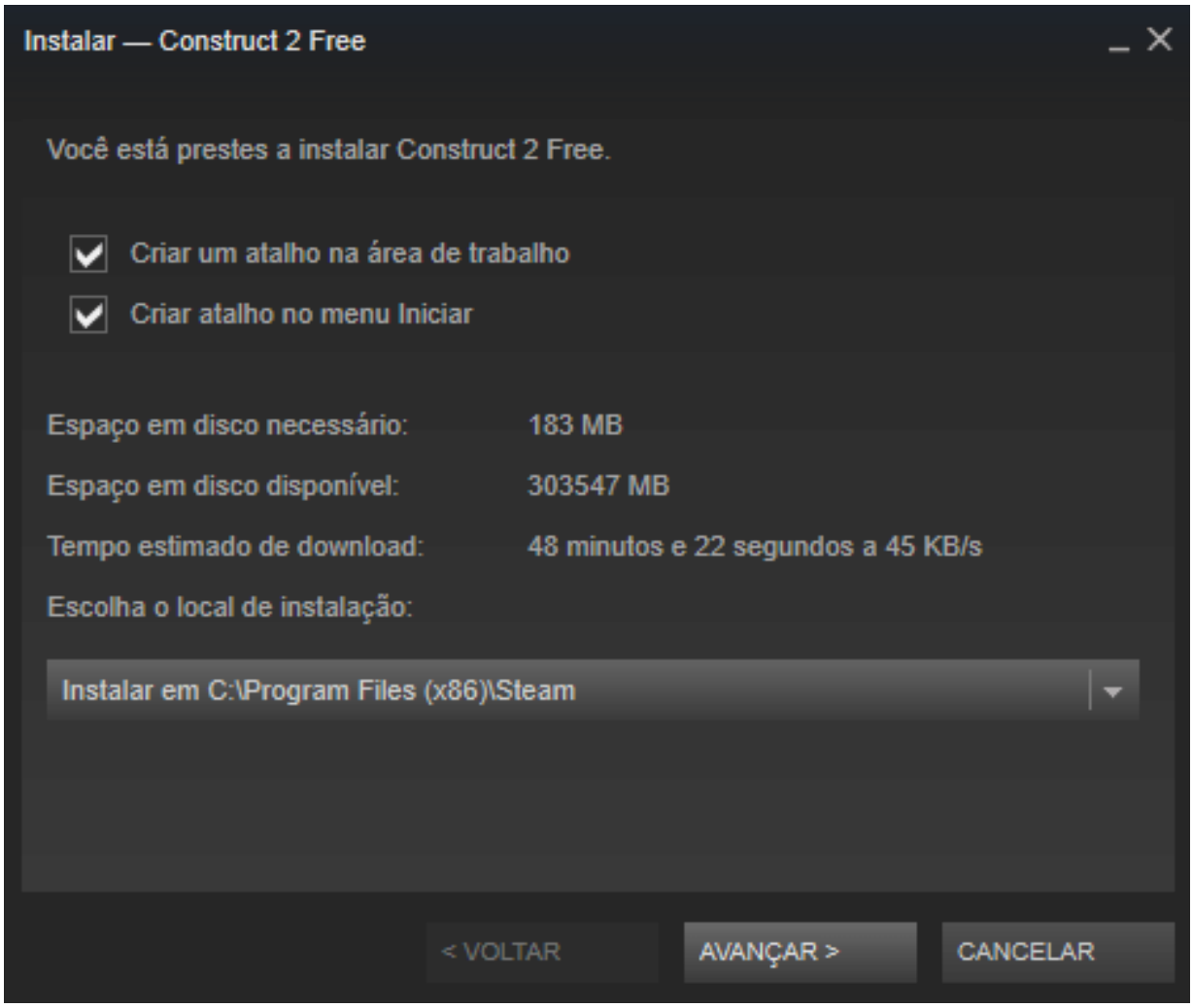

Instalar — Construct 2 Free
Você está prestes a instalar Construct 2 Free.
Criar um atalho na área de trabalho
Criar atalho no menu Iniciar
Espaço em disco necessário: 183 MB
Espaço em disco disponível: 303547 MB
Tempo estimado de download: 48 minutos e 22 segundos a 45 KB/s
Escolha o local de instalação:
Instalar em C:\Program Files (x86)\Steam
< VOLTAR
AVANÇAR >
CANCELAR

Figura 1.9: Informações do download

Clique no botão `Avançar`.

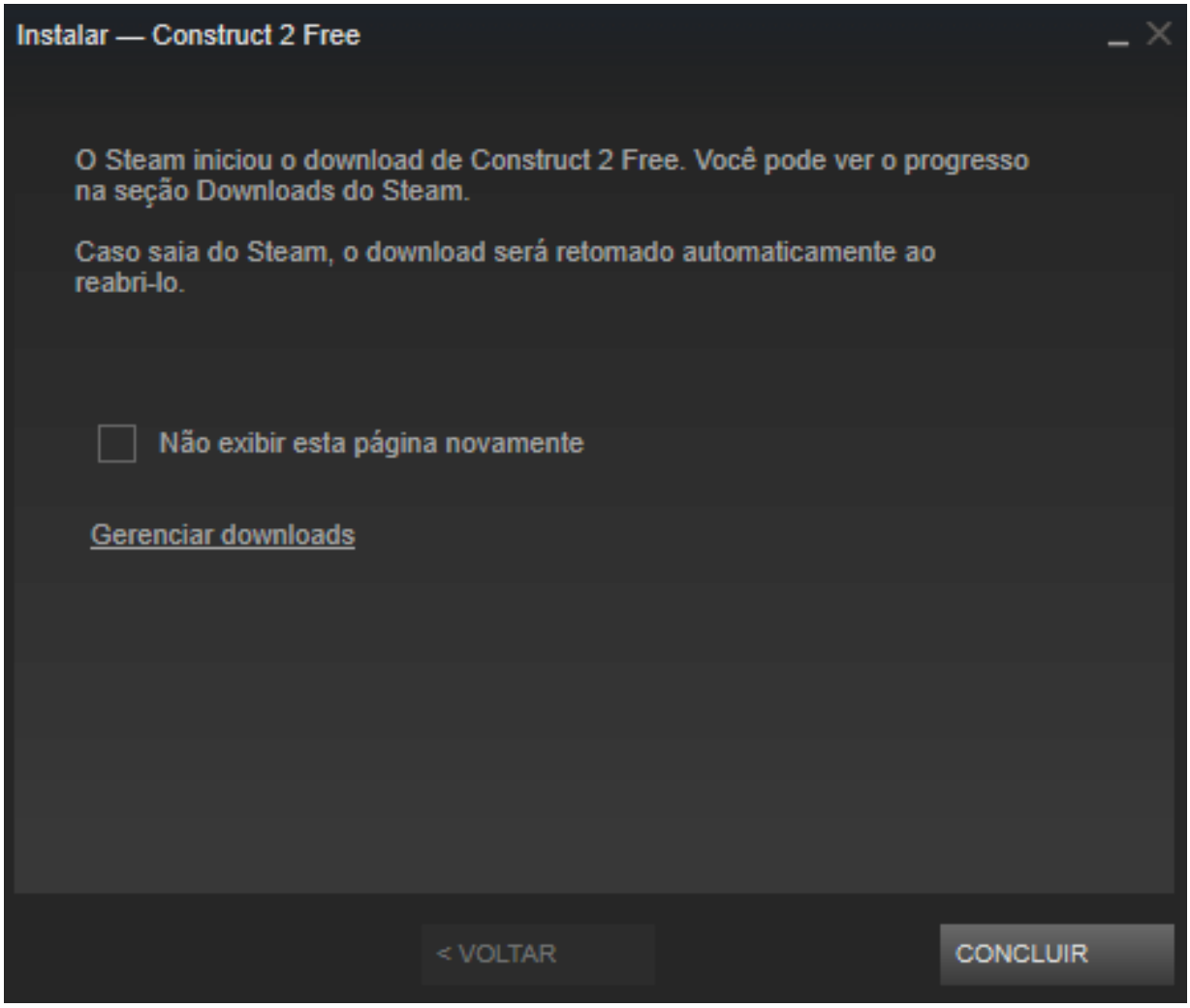

Instalar — Construct 2 Free
O Steam iniciou o download de Construct 2 Free. Você pode ver o progresso na seção Downloads do Steam.
Caso saia do Steam, o download será retomado automaticamente ao reabri-lo.
Não exibir esta página novamente
Gerenciar downloads
< VOLTAR
CONCLUIR

Figura 1.10: Instalar

Na aba dessa imagem, basta clicar em `Concluir`, e o processo do download vai começar. Uma vez finalizado, estará pronto para uso!

1.4 Plataformas que o Construct 2 desenvolve

Um dos grandes pontos altos da ferramenta, além da sua facilidade, é a possibilidade de programar o jogo somente uma vez e poder exportá-lo para uma grande quantidade de plataformas diferentes. Isso deve-se ao fato de que todos os jogos são criados e exportados

em JavaScript por debaixo dos panos, por meio da tecnologia oferecida pelo Apache Cordova (https://cordova.apache.org/), um projeto de código aberto mantido pela Apache Software Foundation. Ele nos permite construir aplicações desktop e mobile usando tecnologias web, como HTML5, CSS e JavaScript.

Como já foi ressaltado, na versão gratuita só é possível exportar para HTML5 e posteriormente hospedar em um domínio HTTPS. Porém, as versões *Personal* e *Business* permitem exportar jogos para os seguintes dispositivos e sistemas:

- Web (HTML5);
- Wii U;
- iOS;
- Android;
- Windows Phone 8;
- Blackberry 10;
- Windows;
- Mac OS X;
- Linux;
- Firefox Marketplace;
- Tizen;
- Facebook;
- Chrome Web Store;
- Amazon Appstore.

Nos próximos capítulos, vamos aprender como exportamos os jogos para essas plataformas.

Jogos famosos já criados

Há milhares de jogos que foram criados com a plataforma desde o seu lançamento. Somente no site da Scirra Arcade (https://www.scirra.com/arcade/), há mais de 12.500 jogos publicados. Dentre eles, ficam em destaque os que possuem as melhores classificações entre os usuários.

Veja alguns dos sucessos criados com o Construct 2:

Título	Informações
There Is No Game	Criado por: Kamizoto Classificação: 4/5 estrelas Jogadores/Plays: 376.628 / 747.013 Data de publicação: 26/06/2016
Tag	Criado por: Joeriri Classificação: 4/5 estrelas Jogadores/Plays: 22.285 / 106.225 Data de publicação: 28/05/2015
Accelo alone	Criado por: Baursak Classificação: 4/5 estrelas Jogadores/Plays: 12.354 / 73.923 Data de publicação: 13/07/2015
Dreams and Reality	Criado por: AlceX Classificação: 4/5 estrelas Jogadores/Plays: 9.539 / 16.297 Data de publicação: 19/06/2016

Muitos desses jogos ganharam prêmios importantes em competições, e também foram publicados em outras plataformas além da web, como plataformas mobile ou até mesmo no Wii U.

1.5 Scirra Store

A Scirra, desenvolvedora do Construct 2, também mantém uma loja oficial para a plataforma, a Scirra Store. Nela é possível encontrar não só as licenças e os jogos prontos, mas também vários outros recursos:

- Arquivos de imagens (personagens, cenários etc.);

- Áudios (músicas e sons de efeitos);
- Templates (jogos pré-construídos);
- Tilemaps (imagens para construção de cenários);
- Fontes customizadas (para textos e menus);
- E-books (livros com documentação e dicas);
- Bundles (pacotes com vários recursos com preços diferenciados).

Vale lembrar que todos os produtos podem ser pagos com a moeda local (R$), mesmo que o site esteja em inglês. Porém, também há muitos produtos gratuitos que podem ser usados. Só é preciso ficar atento à licença de uso de cada um, já que alguns desenvolvedores não permitem que você utilize os seus recursos para fins lucrativos.

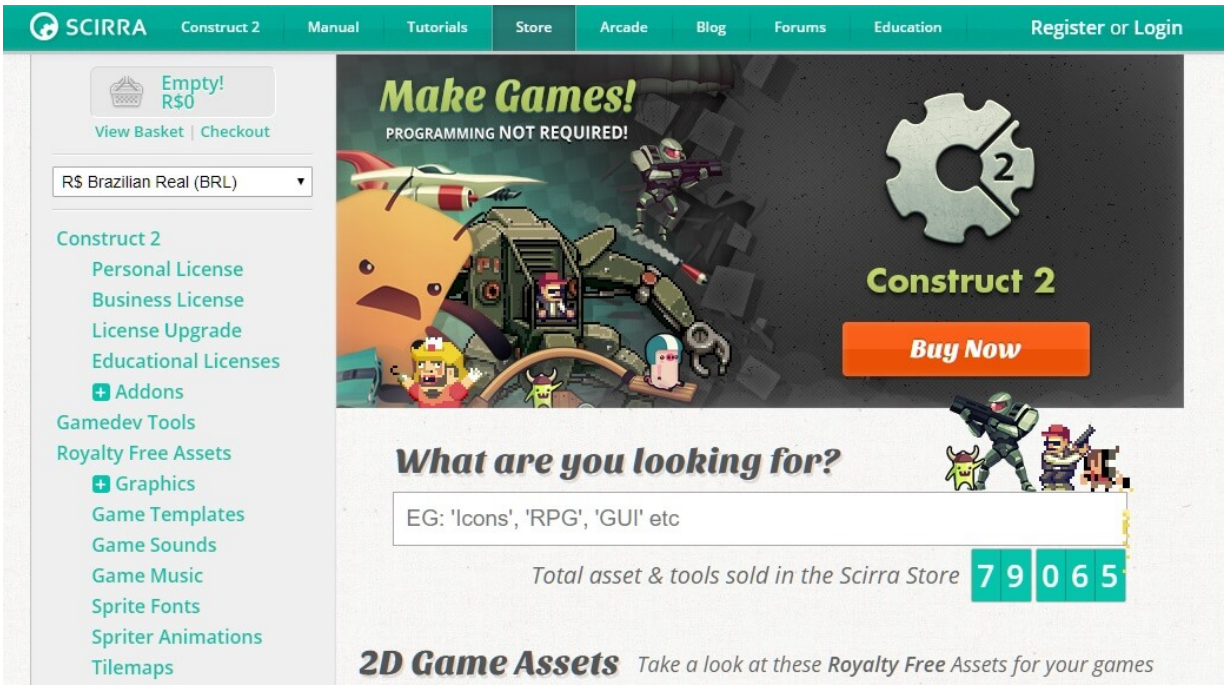


Figura 1.11: Loja da Scirra

Outro ponto que vale comentar é que qualquer usuário pode subir conteúdo (jogos ou trabalhos artísticos) para a loja da Scirra. Desta forma, é possível distribuir o seu trabalho para toda a comunidade do Construct 2. A loja possui uma série de vantagens, como:

- Visibilidade mundial para toda a comunidade do Construct 2;

- Suporte para várias moedas (real, dólar, euro etc.);
- Facilidade para subir os produtos e uma boa documentação;
- Processo simples de venda e recebimento;
- Boas práticas de SEO (*Search Engine Optimization*), facilitando que o seu produto seja encontrado nos mecanismos de busca.

Para comprar produtos, basta criar uma conta gratuita e assinar os termos de contrato da loja. Todo os itens comprados sempre ficarão acessíveis por lá, sendo possível baixá-los novamente caso você os perca.

Mas se você quiser vender as suas produções, será necessário pagar uma taxa de licenciamento de venda, que hoje custa aproximadamente R$ 60,00. Feito isso, todo produto que você submeter será avaliado por eles e liberado caso esteja de acordo com a descrição enviada. Tenha em mente que, para cada venda, a Scirra retém uma taxa de 30% em cima do valor da compra.

1.6 Onde conseguir ajuda

O Construct 2 foi desenvolvido pensando nas pessoas que têm curiosidade e desejo de construir seus próprios jogos, mas não têm uma base de conhecimento tecnológico para isso – o que é exigido em outras plataformas famosas, como o Unity e o Unreal. A princípio, realmente não é necessário saber nenhuma linguagem de programação nem design de games para começar a trabalhar. Como veremos ao decorrer do livro, com poucos cliques, é possível criar seus primeiros jogos.

Entretanto, vale apontar que, com a evolução natural dos projetos, mais conhecimentos tornam-se necessários para construir um jogo cada vez melhor, tanto visualmente quanto tecnicamente (otimização de recursos e esforço computacional). Mas isso faz parte da evolução natural. Independente do nível de afinidade com a

ferramenta, há uma verdadeira infinidade de sites e fóruns na internet que contêm tutoriais e manuais da ferramenta e podem ser consultados gratuitamente.

Vejamos a seguir os principais canais oficiais:

- **Scirra Tutorials** (https://www.scirra.com/tutorials/top) – Este é o espaço oficial de tutoriais da Scirra. Todos os usuários cadastrados podem criar e explorar os tutoriais disponíveis. A maioria deles acompanha o código-fonte e está dividida por dificuldade, assunto e língua. Infelizmente, os artigos em nossa língua ainda são poucos por lá, sendo a maior parte em inglês. Há tutoriais para diversos tópicos diferentes: publicações, física, dicas e truques, performance, som, inteligência artificial, gráficos etc.
- **Scirra Forum** (https://www.scirra.com/forum/) – Este é o espaço reservado para que as pessoas tirem suas dúvidas e discutam suas ideias com outros desenvolvedores de jogos ao redor do planeta. O fórum é bastante movimentado e os participantes são bem receptivos.

Fora estes dois canais oficiais, há muitos outros brasileiros e estrangeiros no YouTube que dão dicas e tiram dúvidas sobre as ferramentas. Uma outra possibilidade é participar de grupos do Facebook sobre a plataforma e sobre desenvolvimento de jogos em geral. Meetups, workshops e grupos de estudos também são uma excelente opção. Seguem algumas recomendações:

- Time to Play (http://timetoplay.com.br/): empresa especializada na produção de jogos digitais. Mantém uma série de artigos, vídeos, recursos e afins para os seus jogos.
- Meetup Construct 2 - SP (https://www.meetup.com/pt-BR/Construct2-Sao-Paulo/): grupo de meetup para reuniões com intuito de discutir ideias sobre a plataforma.
- Grupo Construct 2 Brasil (https://www.facebook.com/groups/Construct2BR/): o maior

grupo do Facebook sobre Construct 2 do país. Tem uma comunidade bastante ativa que está sempre se ajudando.

1.7 Conclusão

Esperamos que agora você tenha uma boa noção de todo o ecossistema do Construct 2: onde é possível obter a ferramenta, suas limitações, como instalá-la em sua máquina, o que é necessário para usá-la, onde obter recursos, como vendê-los e como conseguir ajuda.

No próximo capítulo, veremos mais de perto a plataforma e o funcionamento de sua interface, páginas e mais detalhes.

CAPÍTULO 2
Conhecendo a plataforma

O primeiro passo para aprender como funciona qualquer programa de computador é entender a sua interface e saber onde encontrar as ferramentas, o que elas fazem e como configurá-las. Neste capítulo, daremos uma visão geral dos principais aspectos funcionais da interface do Construct 2 para dar um gostinho do poder da ferramenta.

Assim, poderemos começar a criação do nosso projeto, desenvolvido ao decorrer do livro. Prontos?

2.1 Start page, a tela de início

Logo após abrir a aplicação, a primeira tela em evidência será a `Start Page`. Esta página é dividida em seções que facilitam a navegação e o acesso a projetos novos ou que já estão em andamento.

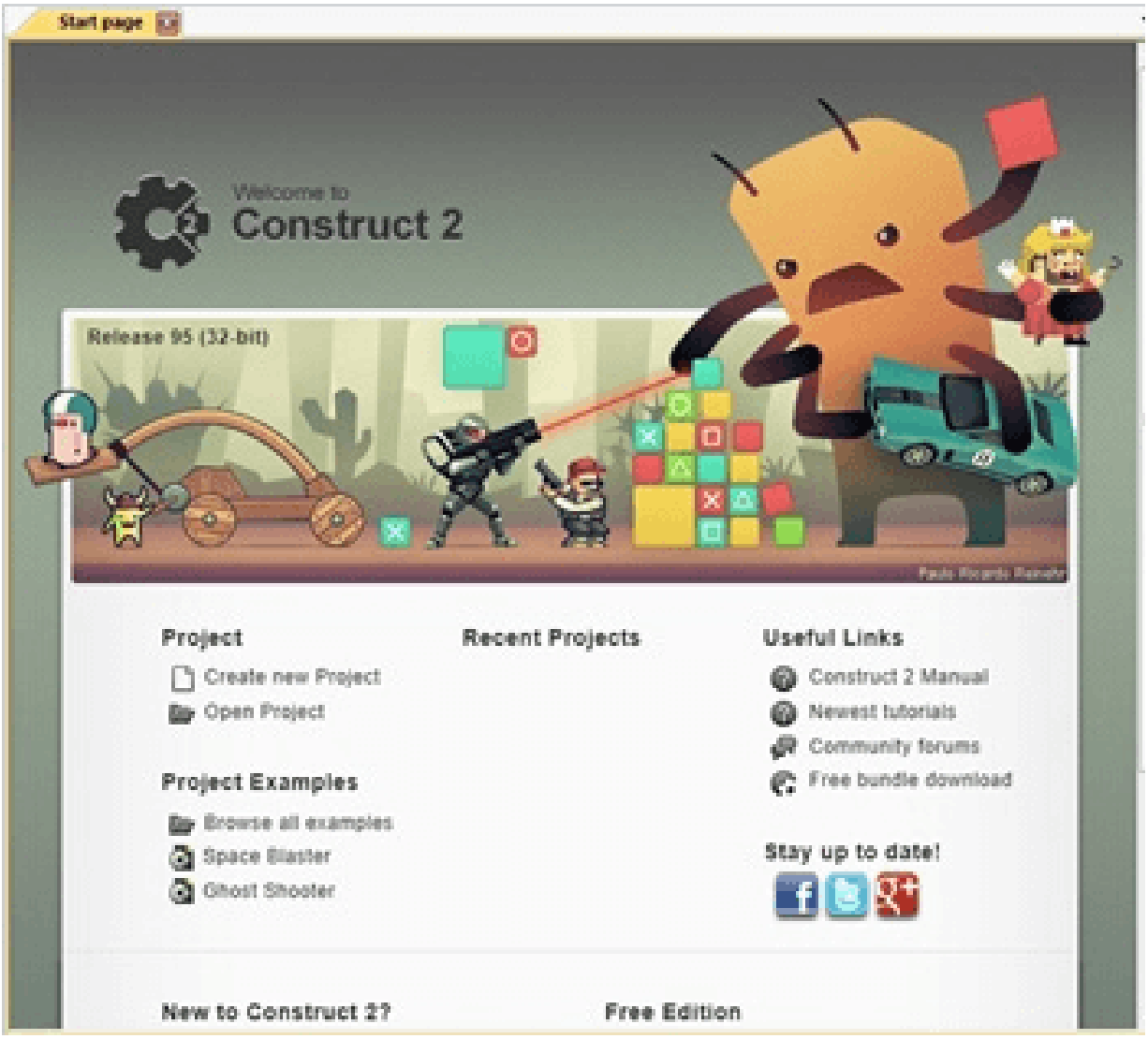


Figura 2.1: Start page

Na primeira seção `Project`, podemos criar ou abrir um novo projeto. Logo abaixo, em `Project Examples`, podemos acessar o código-fonte de uma série de pequenos exemplos de projetos pré-construídos que vêm com a plataforma.

O intuito é que a pessoa use estes modelos para aprender como funciona a ferramenta e como construir diversos tipos de jogos. Na versão que serve de base para este livro, há 92 exemplos. Com eles, é possível aprender como:

- Criar áudios (som de fundo, efeitos sonoros etc.);

- Trabalhar com diversos tipos de movimentação para o personagem;
- Desenvolver um jogo multiplayer;
- Criar partículas (como fogo, água, raio);
- Trabalhar com a física;
- Criar jogos de plataforma;
- Utilizar sombras nos personagens;
- Escutar eventos de toque na tela dentro do jogo;
- Entre outras coisas.

Há muitos exemplos bacanas que devem ser explorados e ajudam bastante na hora em que surge uma dúvida, ou quando a criatividade está em baixa.

Para criar um novo projeto, clique em `New Project` na `Start Page`. Escolha a primeira opção de template vazio `New empty project`, conforme mostra a figura:

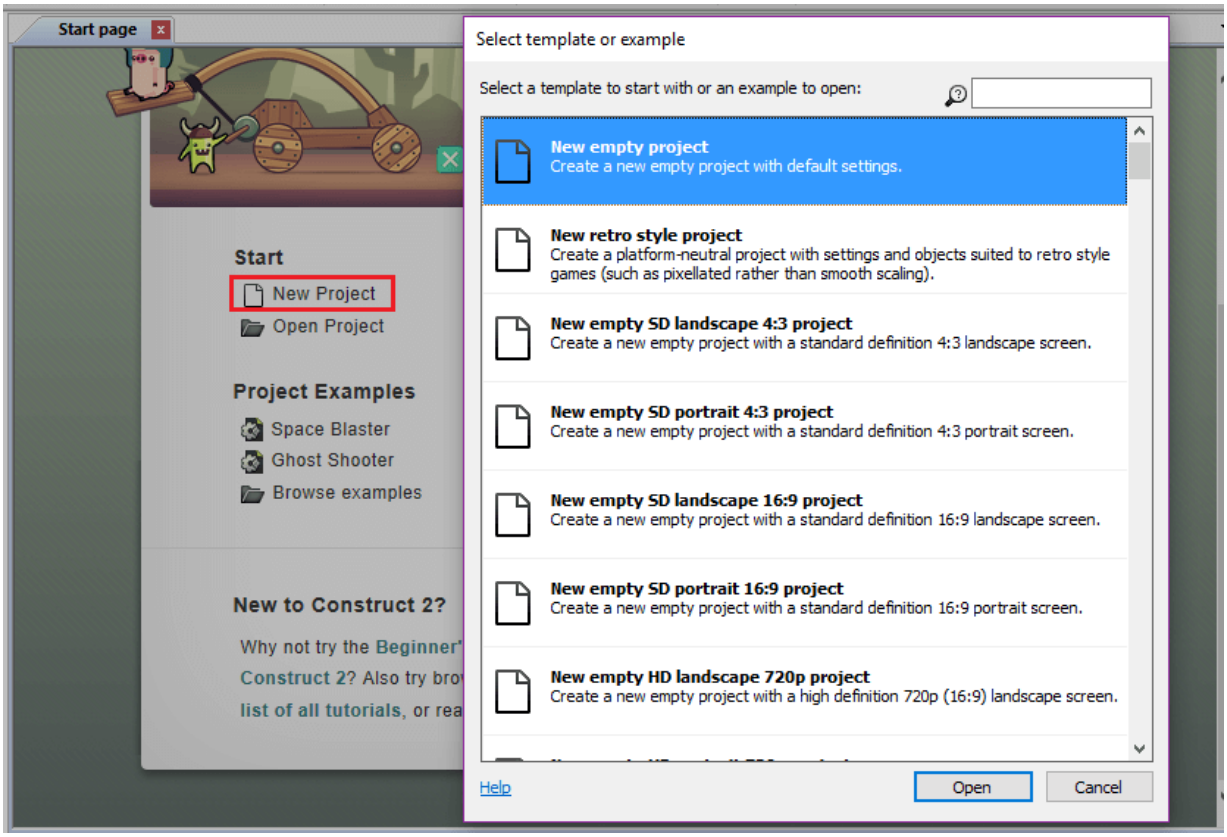

Start page
Start
New Project
Open Project
Project Examples
Space Blaster
Ghost Shooter
Browse examples
New to Construct 2?
Why not try the Beginner'
Construct 2? Also try bro
list of all tutorials, or rea
Select template or example
Select a template to start with or an example to open:
New empty project
Create a new empty project with default settings.
New retro style project
Create a platform-neutral project with settings and objects suited to retro style games (such as pixellated rather than smooth scaling).
New empty SD landscape 4:3 project
Create a new empty project with a standard definition 4:3 landscape screen.
New empty SD portrait 4:3 project
Create a new empty project with a standard definition 4:3 portrait screen.
New empty SD landscape 16:9 project
Create a new empty project with a standard definition 16:9 landscape screen.
New empty SD portrait 16:9 project
Create a new empty project with a standard definition 16:9 portrait screen.
New empty HD landscape 720p project
Create a new empty project with a high definition 720p (16:9) landscape screen.
Help
Open
Cancel

Figura 2.2: Criando um projeto

2.2 Conhecendo a interface

Com o projeto criado, você verá uma página semelhante com a figura:

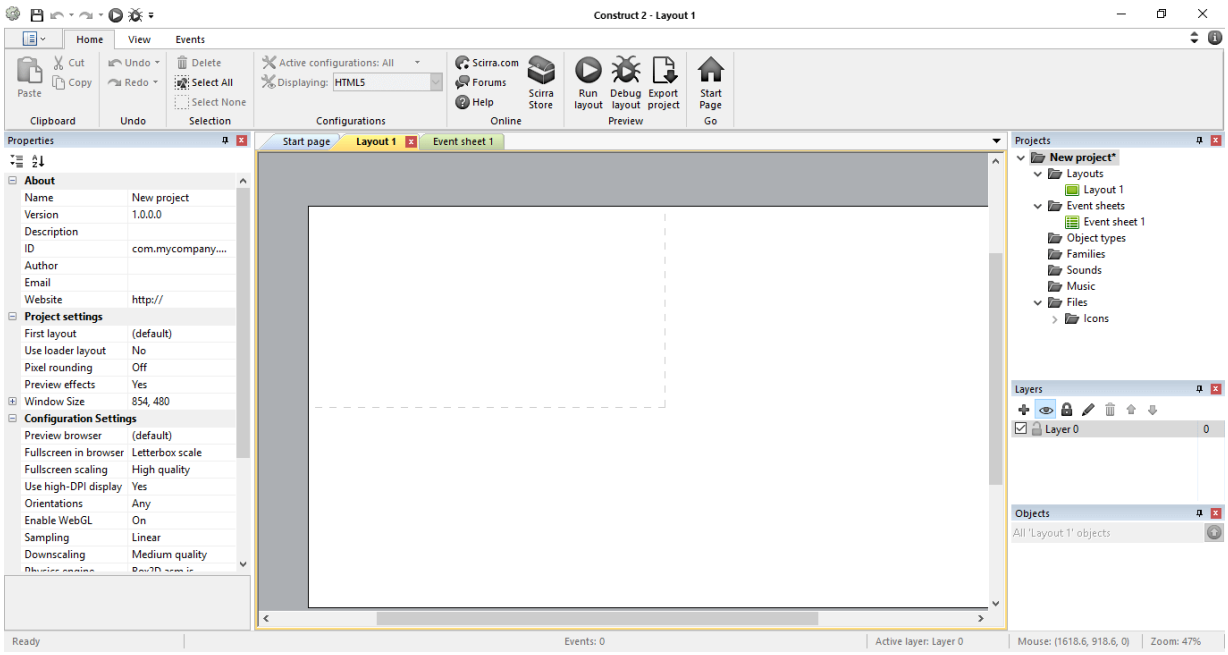


Figura 2.3: Interface

O Construct está em constante evolução

É importante mencionar que, quando você abrir o Construct 2, é possível que a interface esteja um pouco diferente do que a mostrada neste livro. Uma das barras pode não estar no mesmo lugar ou sequer nem aparecer. Isso acontece porque elas são móveis e podem ser posicionadas do jeito que o usuário desejar. Além disso, a plataforma contém constantes atualizações, e uma delas pode acabar alterando o seu visual. Mas é sempre possível voltar às opções padrão em `File > Preferences > Reset dialogs`.

Ao lado direito, podemos ver que foram criadas algumas pastas automaticamente:

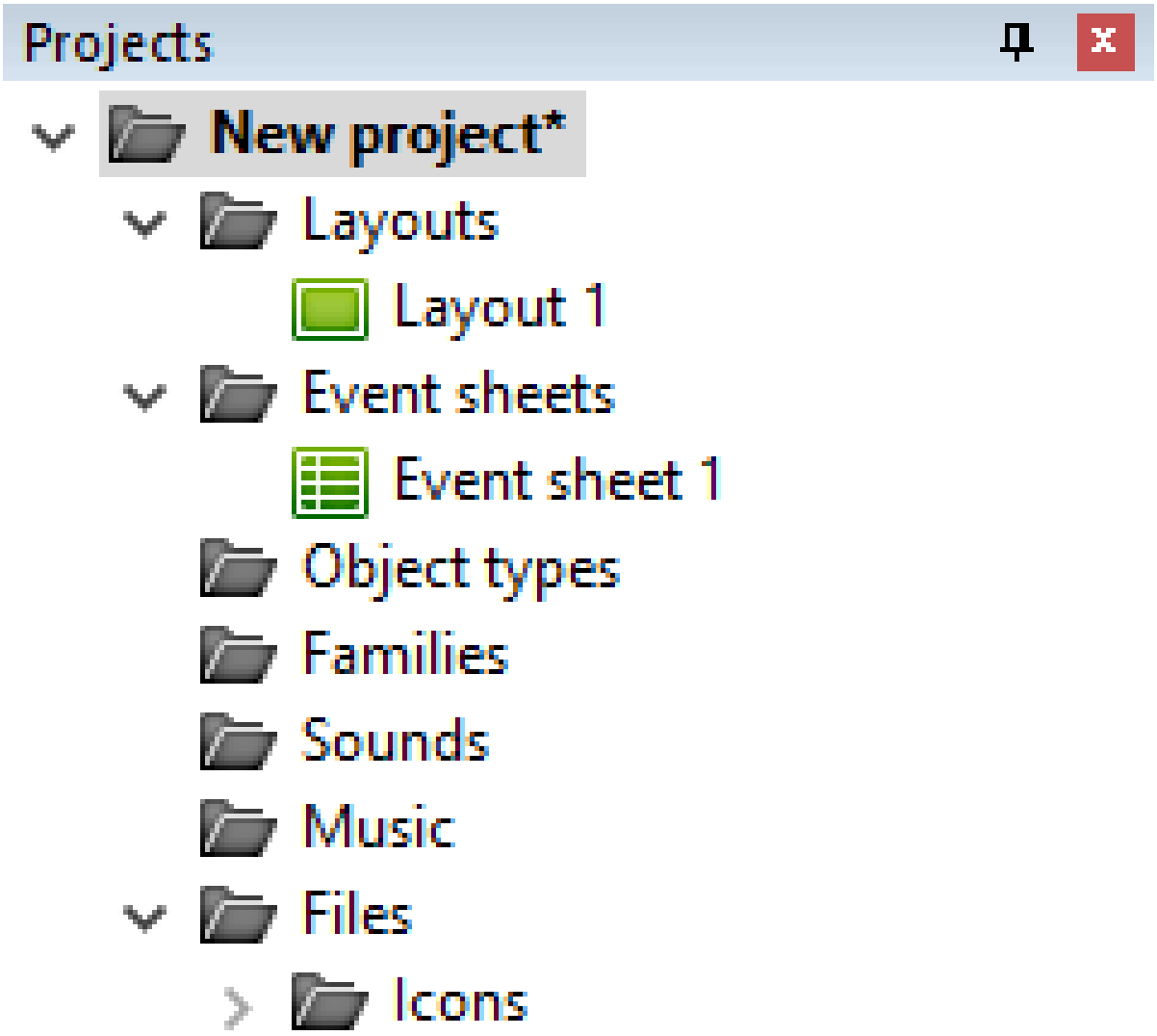


Figura 2.4: Aba Projects

Uma das páginas criadas fica na pasta `Layouts` e é chamada de `Layout 1`. O *layout* é uma peça fundamental do funcionamento da plataforma. É nele onde serão inseridos os sons, as imagens e todos os demais objetos do jogo, ou seja, é onde a nossa fase será criada. Todos os jogos necessariamente possuem ao menos um layout.

A outra página também é outra peça fundamental e é criada na pasta `Events`, com o nome `Event sheet 1`. É nas páginas do tipo `Event Sheet` que fica toda a programação de eventos do jogo. A

Layout 1 é associada à Event sheet 1 automaticamente, no entanto, com a criação de outras páginas, é possível desfazer essa associação através da Barra de Propriedades .

O retângulo com linhas pontilhadas no Layout 1 delimita até onde o jogador poderá enxergar. É a área visível que aparece na tela do jogo, chamada de *Window*. Porém, é importante salientar que um layout não precisa ser necessariamente do mesmo tamanho definido na Window, já que, dependendo do jogo – como os jogos de plataforma –, a fase sempre será maior do que o jogador consegue ver na tela.

Para o nosso exemplo, vamos definir um tamanho pequeno no Layout Size e no Window Size para 640 por 480 (pixels), no menu de propriedades (Properties) ao lado esquerdo da interface. Como o próprio nome sugere, essa aba mostra as propriedades do objeto que estiver selecionado. Entretanto, se nada estiver selecionado, ela mostrará só as propriedades do layout, onde será possível nomear, modificar o tamanho, entre outras coisas.

Se você selecionar a pasta raiz do seu projeto no menu lateral, serão exibidas propriedades sobre o seu projeto, como: nome, versão, descrição, autor etc. Você vai perceber que, com esse tamanho, as linhas pontilhadas do Layout 1 vão sumir, portanto, o jogador enxergará o layout inteiro do jogo.

Dentro do retângulo pontilhado, vamos adicionar um objeto que será a base para criação do nosso personagem de plataforma. Para isso, clique com o botão direito do mouse e selecione a opção Insert New Object (Inserir novo objeto). Isso abrirá uma janela com os plugins internos do Construct 2, conforme mostra a figura:

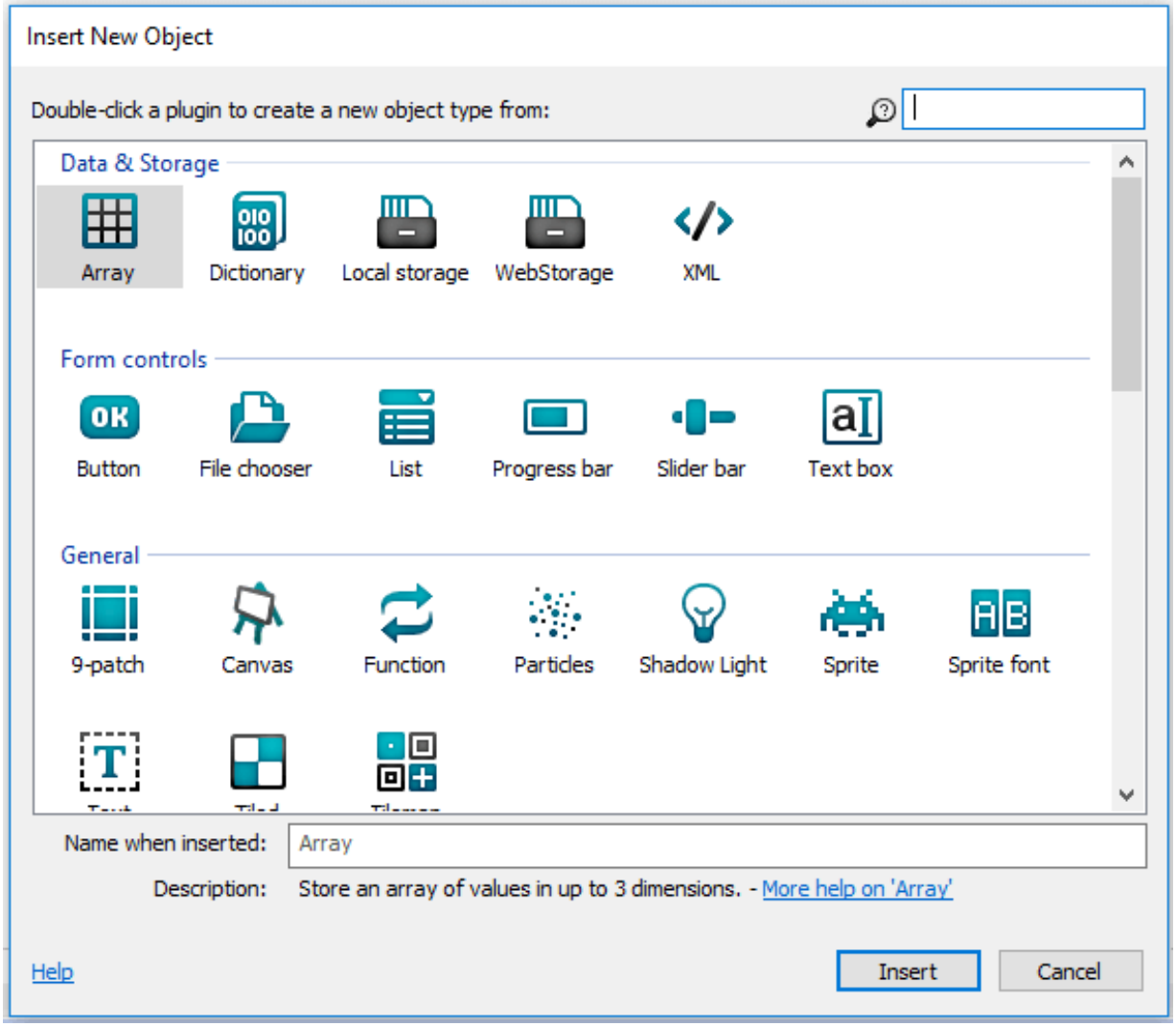


Figura 2.5: Inserindo objetos

Vamos escolher o plugin de `Sprite`, que consiste em uma simples imagem que pode ser animada e usada nos elementos visuais de um jogo. Vamos nomeá-la de `Jogador`. Feito isso, coloque-a dentro do `Layout 1`.

Com isso, o editor de imagens do Construct 2 abrirá. Escolha a ferramenta de pintura `Fill` e uma cor para pintar a `Sprite`. Escolha também a ferramenta `Resize` para ajustar o seu tamanho para 32 por 68 pixels, para simbolizar um personagem de plataforma, conforme exemplificado:

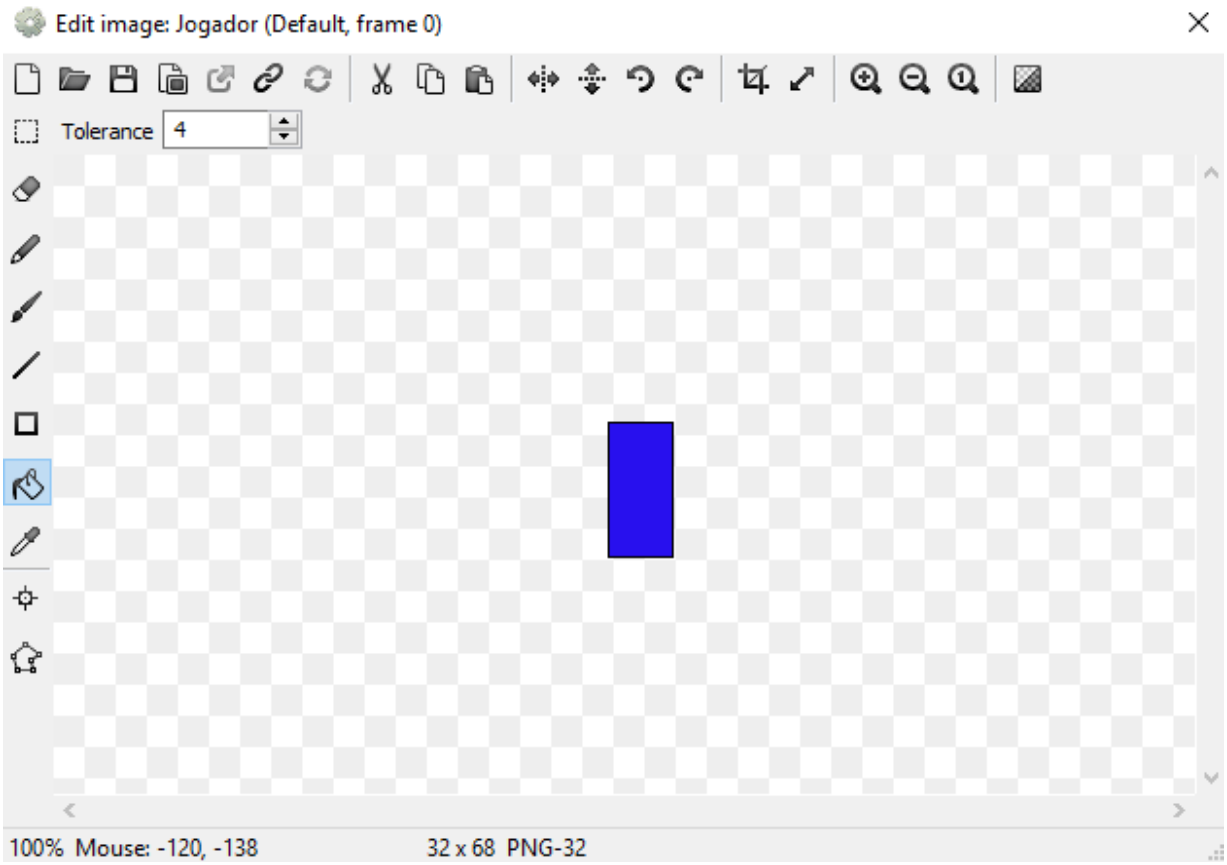


Figura 2.6: Editor de imagens

Para dar "vida" ao personagem, vamos adicionar uma `Behavior` (comportamento). As `Behaviors` são uma das funcionalidades mais legais do Construct 2, pois nos permitem adicionar comportamentos específicos para os nossos objetos. Existem vários tipos de comportamentos disponíveis, desde carros, plataformas, sólidos, física etc.

As `Behaviors` nos ajudam a dar forma aos nossos personagens, sem a necessidade de codificar nada. Vamos utilizá-las bastante na construção do nosso projeto.

Para adicionar uma `Behavior` , clique sobre o objeto que acabamos de inserir e, ao lado esquerdo nas suas propriedades, escolha a opção de `Behavior` . Na janela que abrir, clique em `+` (mais) e dê um

duplo clique para adicionar a `Behavior Platform` (plataforma), conforme mostra a figura adiante.

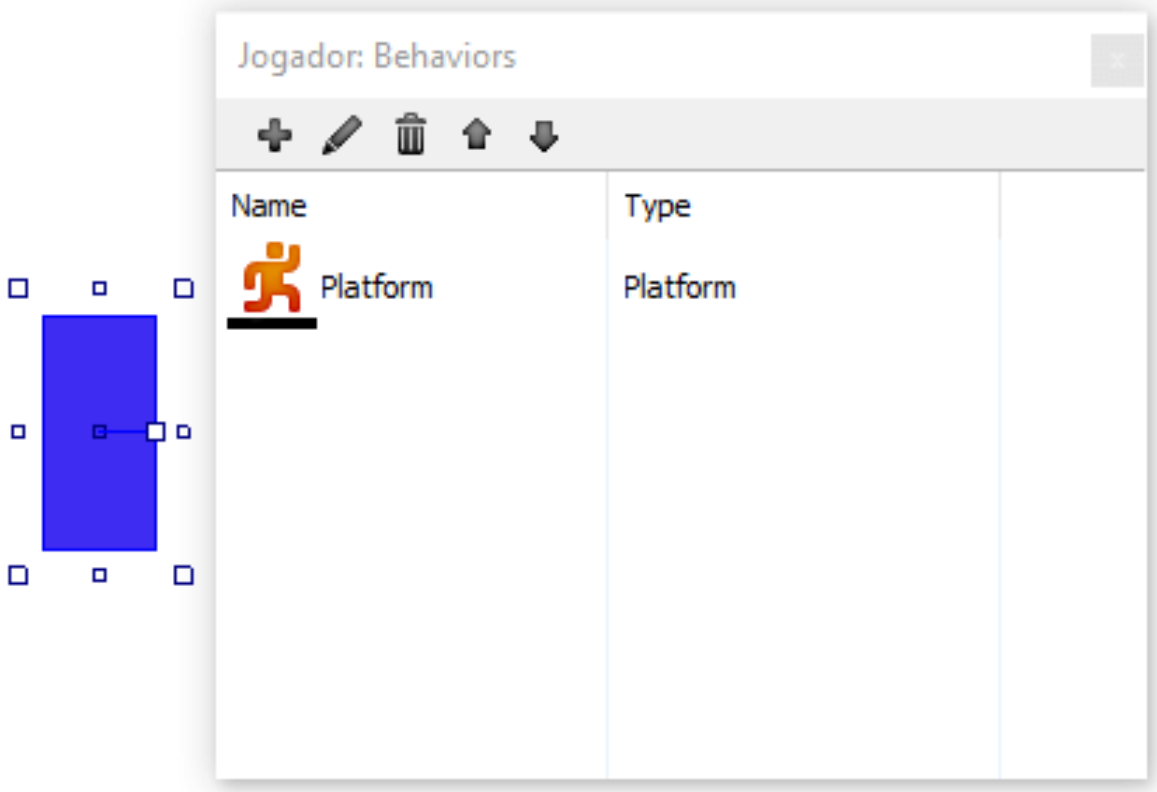


Figura 2.7: Adicionando uma Behavior

O comportamento de plataforma faz com que o objeto seja movimentado pelas setas do teclado nas quatro direções (cima, baixo, direita e esquerda). Em suas propriedades, podemos configurar a velocidade, a aceleração, a força do pulo, a ação da gravidade, entra outras características. Por ora, não vamos modificar nada.

Antes de testar o personagem, vamos adicionar uma `Sprite` que vai representar o chão por onde ele andará. Para isso, vamos adicionar um plugin na aba de inserção de objetos chamado `Tiled Background` – a tradução mais clara para o português seria algo como "Fundo de

mosaico". Quando o editor de imagens abrir, desenhe algo para ser o chão.

Os objetos de `Tiled Backgroud` exibem uma imagem em um padrão de repetição, ou seja, você pode aumentar o seu tamanho infinitamente, porque ela vai se repetindo. Modifique o tamanho de forma que represente um chão e fique parecido com a figura:

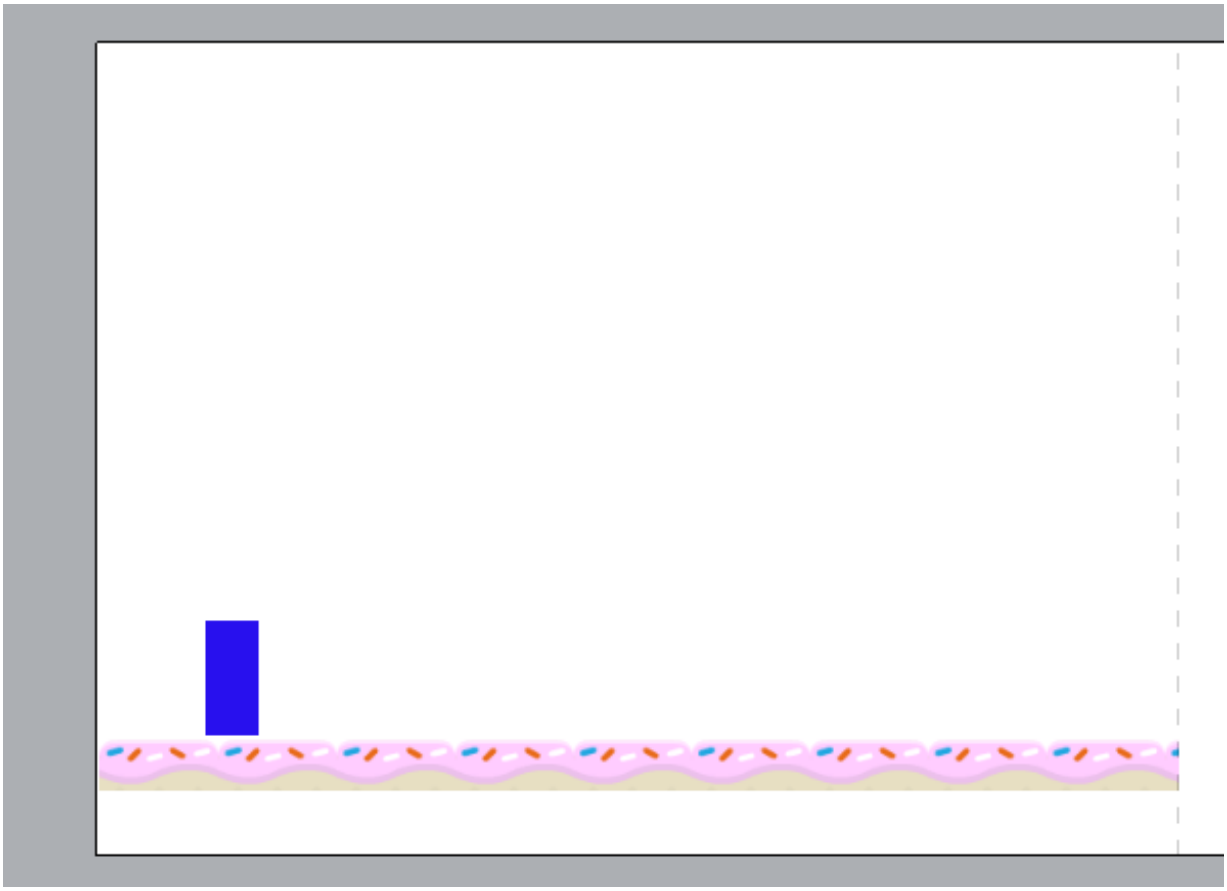


Figura 2.8: Adicionando o chão

Para dar o efeito de chão ao nosso objeto, temos de inserir o comportamento de `Solid` (sólido). Desta maneira, a colisão entre o nosso personagem e o chão é habilitada, fazendo com que o nosso jogador não atravesse o chão, mas, em vez disso, permaneça em cima dele.

Podemos utilizar esse comportamento para criar chão, muros, plataformas móveis ou qualquer tipo de objeto intransponível. Da mesma forma que adicionamos a `Behavior` de plataforma ao nosso personagem, clique no objeto e adicione a `Solid` .

Agora que já temos o nosso personagem e o nosso chão, vamos testar.

2.3 Testes

Agora já podemos testar como nosso jogo está ficando. Primeiramente, escolha o navegador que ele será visualizado usando as configurações do navegador de visualização (`Preview browser`) nas propriedades do projeto ao lado esquerdo da tela:

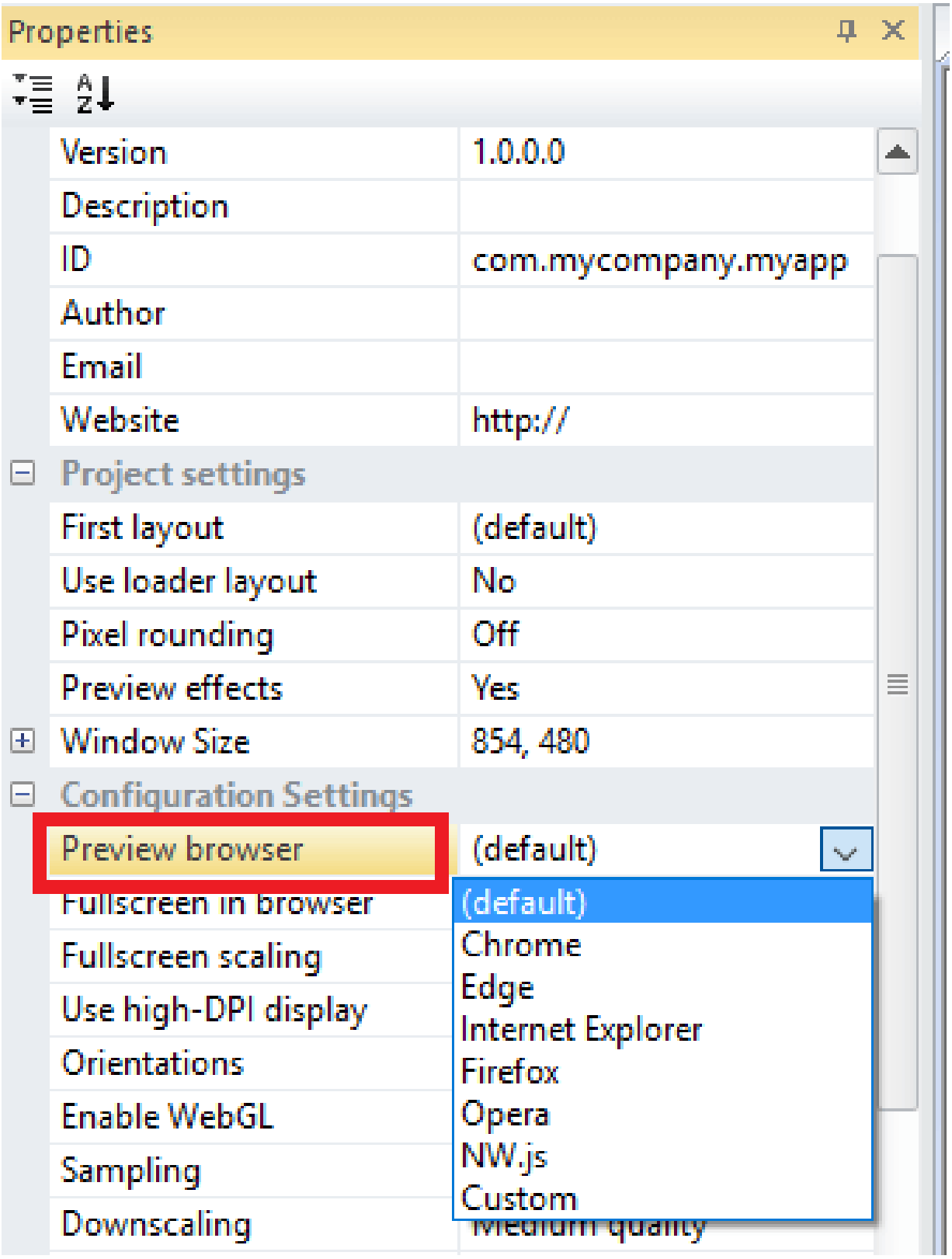


Figura 2.9: Escolhendo o navegador

Depois de escolher o navegador de sua preferência, vá até a barra superior na aba `Home`, conforme mostra a figura, e escolha a opção de `Run layout`.

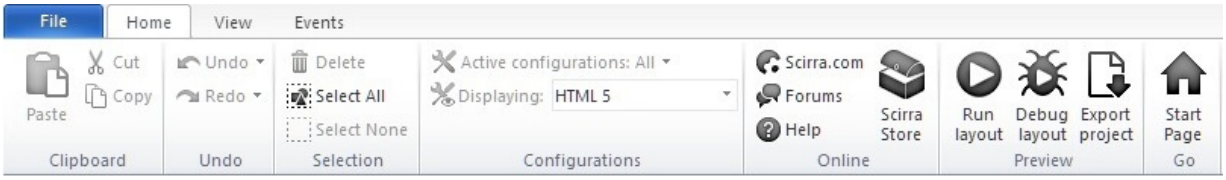


Figura 2.10: Aba Home

Por padrão, o Construct 2 vai subir um servidor local no seu computador, na porta 5000, com o seu jogo. Então, o navegador abrirá o `layout` do seu projeto, conforme mostra a figura a seguir. Use as setas do teclado para movimentar o jogador; você vai perceber que ele já contém os movimentos básicos sem nenhuma programação! Impressionante, não?

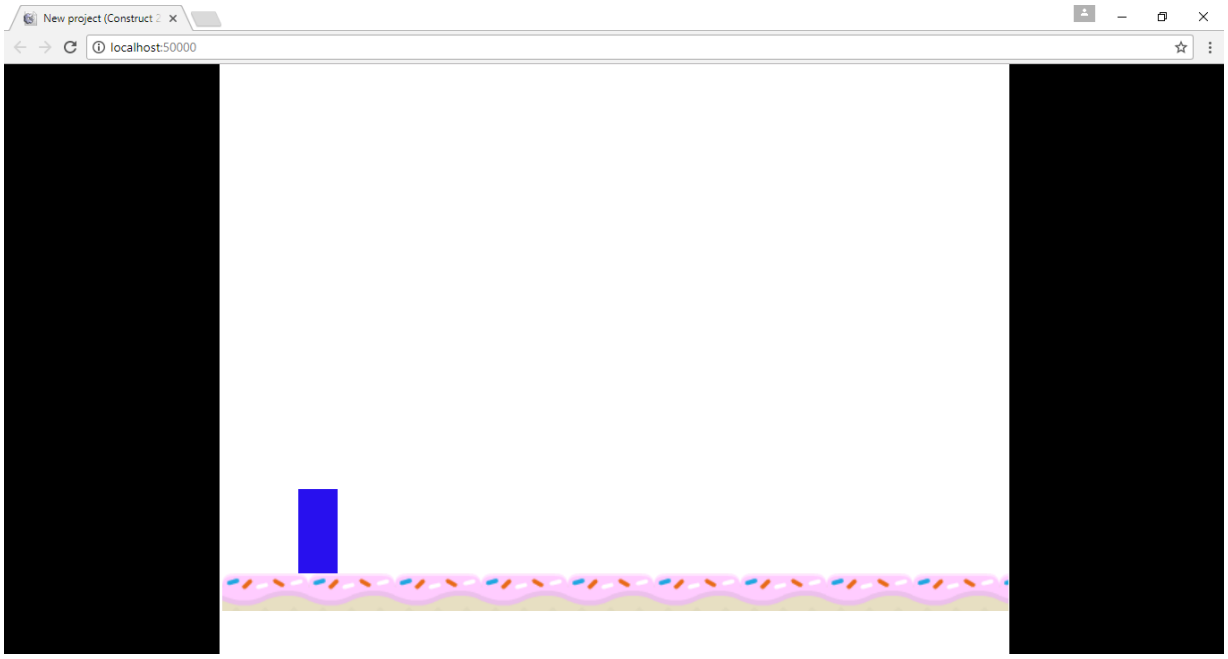


Figura 2.11: Testando o seu jogo

O projeto inteiro é recarregado toda vez que você aperta o `Run layout`. Mas é possível habilitar a visualização contínua, assim ainda é possível jogar durante a visualização no navegador, e podemos

ver a projeção de níveis e pensar em como interagir com eles. Ela mantém o estado atual do seu jogo e só modifica o que foi alterado na programação em vez de carregar tudo de novo.

Para isso, basta pressionar o `Shift` quando clicar em `Run layout`. Você notará que a URL aberta no navegador estará acompanhada com `?continuous`.

Esperamos que você tenha aprendido o básico de como criar um novo projeto, o que são os layouts, como inserir objetos neles, alterar suas propriedades, adicionar `Behaviors` e testar o jogo. Vamos explorar com mais detalhes cada um destes tópicos ao decorrer do livro. No próximo capítulo, daremos início ao nosso projeto.

Capítulo 3
Estrutura inicial

Agora que temos uma noção básica do funcionamento do Construct 2, estamos prontos para seguir em frente. Neste capítulo, vamos começar a criação do nosso jogo de plataforma que se estenderá até o final das explicações.

Vamos ensinar como criar os elementos para que o jogo funcione perfeitamente, mas lhe daremos liberdade para acrescentar/remover objetos e deixar as fases do jogo com a sua cara. Recomendamos que você já baixe no seu computador a pasta de recursos que foi apresentada na introdução, em https://github.com/timetoplaybr/candy-world. Ela contém grande parte dos elementos visuais que vamos usar ao decorrer do desenvolvimento.

3.1 Criando e salvando o projeto

Primeiramente, vamos criar um novo projeto com um template vazio, como fizemos no capítulo anterior. Crie o projeto e salve-o clicando no menu `File > Save As Single File...`. Como é a primeira vez que salvamos o jogo, o sistema vai pedir para você nomear o projeto e selecionar a pasta onde ele será salvo.

Selecione a pasta de sua preferência e salve o jogo com o nome `Candy World`. O jogo será gravado no formato `*.capx`, que significa *Construct 2 compressed project*. Este formato é proprietário da plataforma e compacta todos os nossos recursos neste único arquivo.

Caso você queira manter a estrutura de pastas do seu projeto, é possível salvá-lo no menu `File > Save As Project...`. O Construct 2

então salvará todas as pastas separadamente. Em qualquer um dos casos, uma vez salvo, basta ir em `File > Save` para gravar as novas alterações.

Para manter a integridade e a segurança do seu projeto, o Construct 2 contém configurações padrões de `Autosave` (salvamento automático) e `Backup` (cópia de segurança). Ambas podem ser encontradas e modificadas no menu `File > Preferences`.

Por padrão, é mantido pelo menos um arquivo de backup, e o jogo é gravado automaticamente a cada 20 minutos. A nossa recomendação é que você configure para que seu backup seja armazenado em algum lugar na internet, como no Dropbox, por exemplo. Desta maneira, evitamos perder tudo caso o computador dê algum problema.

Agora que temos nosso projeto criado e salvo, vamos às suas configurações iniciais. No menu de propriedades, preencha as informações do menu `About` (sobre). Neste espaço, devem conter todas as informações sobre o seu jogo, como seu nome, versão, descrição, autor, ID, e-mail e website. Altere o nome para `Candy World` e coloque o restante dos seus dados.

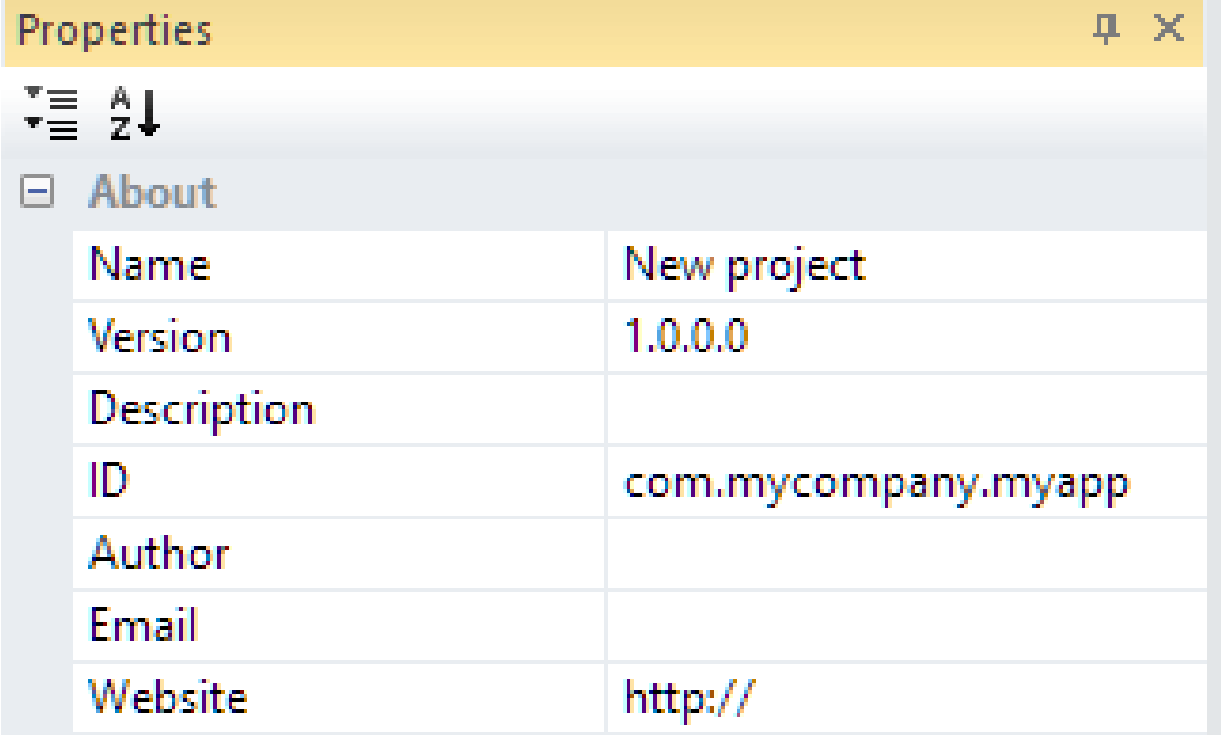


Figura 3.1: Propriedades do projeto

> **O QUE É O ID DE UM JOGO?**
>
> Todo jogo deve conter um identificador único. Normalmente, ele tem o formato `com.<nome_da_empresa>.<nome_do_aplicativo>`. No caso de aplicativos brasileiros, costuma-se ter o `br` antes do `com`.

Agora vamos configurar o tamanho da nossa tela de jogo. Usaremos um tamanho padrão para que ele funcione bem na maior parte dos dispositivos. Defina o tamanho do `Layout Size` para 6000 x 480 pixels, e a `Window Size` para 640 x 480 pixels.

Você notará que o layout ficará muito maior que a área pontilhada. Isso é porque faremos um jogo de plataforma horizontal, então já deixaremos todo o cenário pronto para o nosso personagem atravessar. Por fim, nomeie o `Layout 1` para `Fase1`.

3.2 Adicionando camadas

Ao lado direito da interface, está a `Barra de layers` (Barra de camadas). As camadas têm uma função importantíssima nos jogos: permitem mostrar diferentes grupos de objetos na frente ou atrás dos outros, dando o efeito de profundidade ao nosso jogo, mesmo que ele seja em duas dimensões.

Além disso, elas são essenciais para que possamos trabalhar com interfaces não rolantes (HUDs ou UIs) – em outras palavras, objetos que sempre ficarão no mesmo lugar da tela do jogo, independente da ação do jogador. Este tipo de interface é essencial para colocar informações na tela, como: barra de vida, número de vidas restantes, pontuação, tempo etc.

As camadas sempre pertencem a um layout, podendo ser adicionadas, editadas e excluídas na `Barra de layers`. Para a nossa `Fase1`, vamos adicionar apenas quatro camadas. Para isso, clique no botão `+` para adicionar uma *layer*, e altere seu nome na `Barra de propriedades` ou pressionando a tecla `F2`. Crie as camadas com os nomes: **UI**, **Jogador**, **Principal** e **Fundo**.

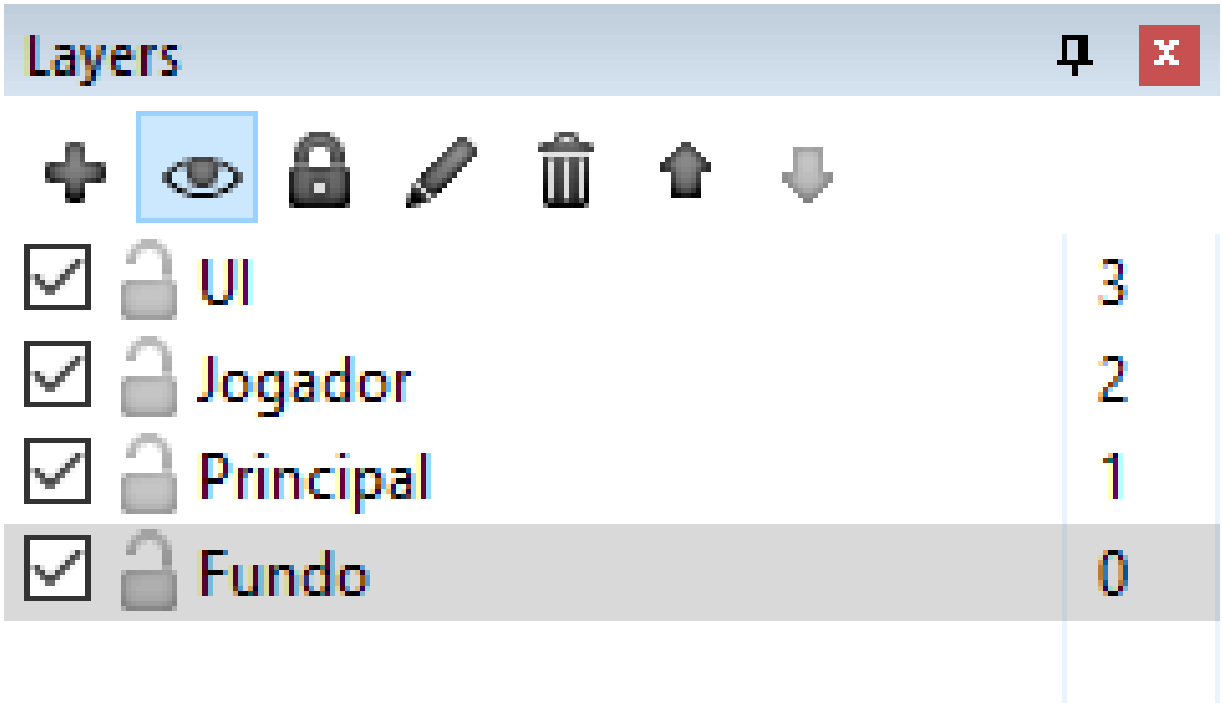


Figura 3.2: Barra de layers

Usaremos estas quatro camadas para organizar os nossos objetos dentro do jogo:

- **Fundo:** aqui colocaremos todos os objetos referentes ao cenário onde o jogo vai se passar – basicamente o plano de fundo e as suas decorações.
- **Principal:** como o nome já indica, será a principal camada no nosso layout. Aqui colocaremos a maior parte dos objetos que vão interagir com o nosso personagem, desde plataformas, portas, colecionáveis até os inimigos.
- **Jogador:** essa camada será dedicada ao nosso personagem e seus inimigos.
- **UI:** esta é a sigla para *User Interface*, que significa interface de usuário. Nesta camada, vamos colocar todos os objetos informativos sobre o jogo, como a pontuação e a quantidade de vidas. Também vamos utilizá-la para colocar a tela de *pause* e a de fim de jogo.

Repare que, ao criar as camadas, todas aparecem com as suas respectivas caixas marcadas, com o cadeado aberto e acompanhadas de um número. Os números indicam a profundidade da camada, de modo que, quanto maior o número, mais à frente ela estará das demais. Por isso que colocamos a camada `Fundo` com `0`, para ela ficar atrás de todas, e a `UI` com `3` para ficar à frente.

Os cadeados servem para "bloquear" uma camada. Quando bloqueada, ela não pode ser alterada. Isso nos ajuda na hora de trabalhar com múltiplas camadas, para evitar mexer em objetos de outra por acidente. Esta é uma técnica bem familiar em programas que costumam trabalhar com elementos gráficos, como o Illustrator e o Photoshop.

Por fim, a caixinha que acompanha os nomes indica se uma camada é visível ou não. Alternar a visibilidade dela nos ajuda a trabalhar com múltiplos objetos no layout. Isso nos permite focar somente na camada que estamos trabalhando no momento.

Ao selecionar uma *layer*, você notará que ela tem algumas propriedades:

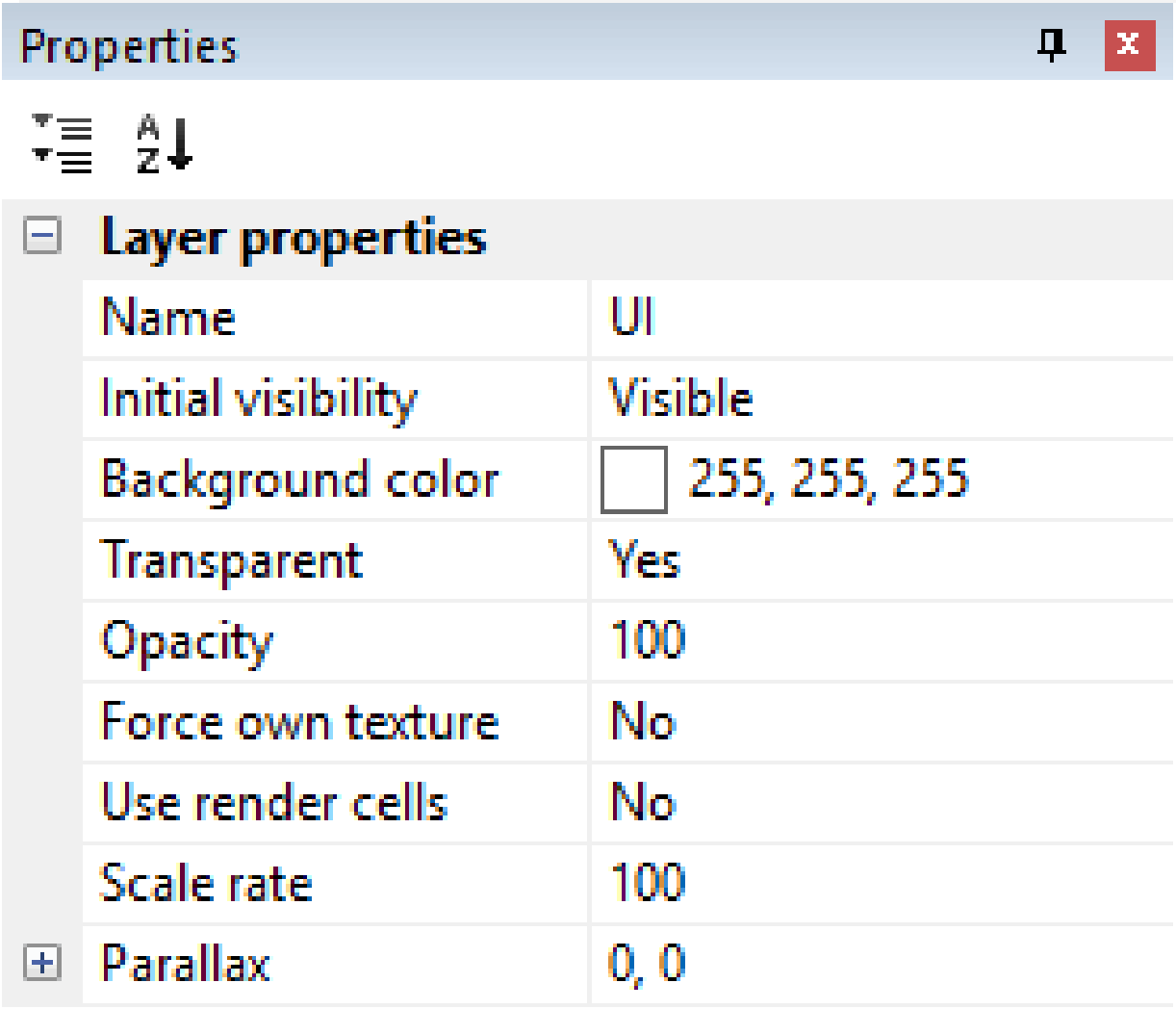


Figura 3.3: Alterando o Parallax

Para o nosso projeto, altere a propriedade `Transparent` (transparente) de todas as camadas para o valor `Yes`. Isso fará com que elas fiquem sem fundo, tornando possível colocar os objetos sobrepostos.

E para que os objetos da camada `UI` fiquem fixos e sempre acompanhem o jogador durante o percurso da fase, altere a propriedade `Parallax` para `0,0`. Caso contrário, os objetos ficarão para trás.

3.3 Adicionando um plano de fundo

Chegou a hora de colocar um plano de fundo para a nossa fase. Para isso, selecione a layer que chamamos de `Fundo` e insira um objeto de `Tiled Background` nela. Os objetos deste tipo são ideais para planos de fundo, porque são otimizados e carregam rapidamente. Por padrão, sempre o usaremos para criar os planos de fundo dos nossos jogos.

Quando o editor de imagens abrir, importe a imagem `fundo-fase1`, que está dentro da pasta `Fundos` na nossa pasta de recursos. Coloque-a no `Layout Fase1` de forma que ocupe todo o tamanho do layout, conforme mostra a figura:

Figura 3.4: Imagem de fundo da fase 1

Repare em como a imagem de fundo se repete perfeitamente durante toda a extensão da fase.

3.4 Criando o cenário do jogo

Criada a nossa fase com um plano de fundo, vamos começar a montar o seu design. Como o nosso projeto será um jogo de plataforma, montaremos nosso cenário usando basicamente quatro itens:

- Chão;
- Plataformas (estáticas e móveis);

- Decorações;
- Barreiras invisíveis.

Para que consigamos trabalhar com eles, é fundamental antes compreender o funcionamento de colisões e pontos de origem e de imagem dentro do Construct 2. Vamos conferi-los.

Origem e pontos de imagem

Sempre que trabalhamos com uma `Sprite` no editor, podemos alterar o seu ponto de origem. Este ponto nada mais é que a origem do objeto. Ele é usado para referenciar a posição dentro do layout e seu centro de rotação. Será esse ponto de rotação que dirá ao Construct 2 como o seu objeto vai agir quando ele for rotacionado.

> **Centro de rotação**
>
> Para entender melhor o que isso significa, pense em como o nosso corpo é composto. Temos dois braços que estão conectados ao restante do nosso corpo por meio dos nossos ombros. Quando você move o seu braço direito, o seu centro de rotação é o seu ombro. Quando você move o seu braço esquerdo, o seu ombro esquerdo é o centro de rotação. Ou seja, o centro de rotação é o ponto fixo do objeto usado como referência para rotacioná-lo.

Essa configuração é feita na ferramenta `Origin and Image Points` (origem e pontos de imagem). Ao clicar nesse botão, é aberto o diálogo de pontos de imagem, o `Image points`. Essa janela permite definir a `Origin` (origem) da imagem e também os seus pontos (pontos arbitrários acessíveis por eventos) em uma imagem:

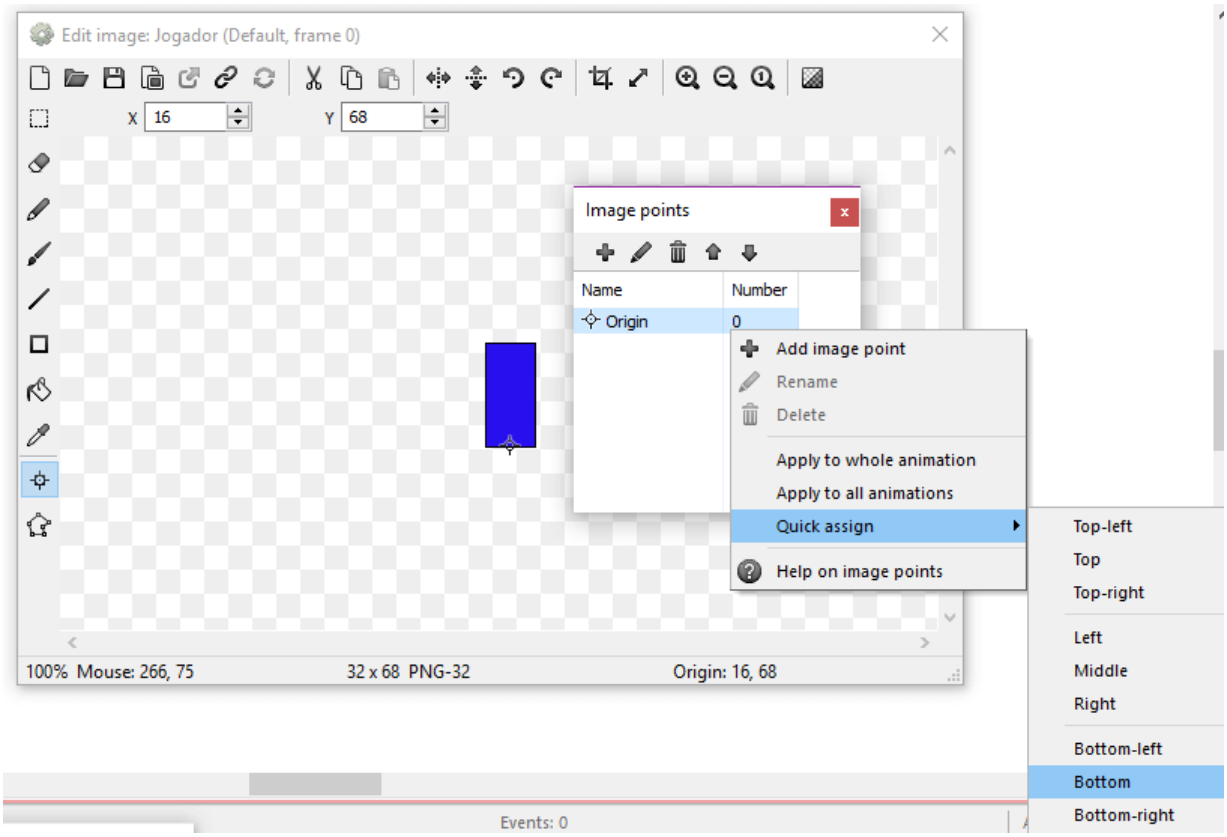


Figura 3.5: Ponto de origem

O ponto de imagem, diferentemente do ponto de origem, tem a função de criar pontos que geram outros objetos. Ou seja, estes são usados como ponto de origem de outros objetos, que saem a partir dele.

O exemplo mais fácil de imaginar como isso é usado é pensar em um jogo de tiro. Ao mesmo tempo que temos o ponto de origem do personagem, também temos um ponto de imagem adicionado à sua arma, pois é de lá que são disparados os tiros. Poderíamos ter uma `Sprite` na qual o personagem carrega uma arma e colocaríamos um ponto de imagem exatamente onde fica a arma, pois é de lá que sairiam as balas.

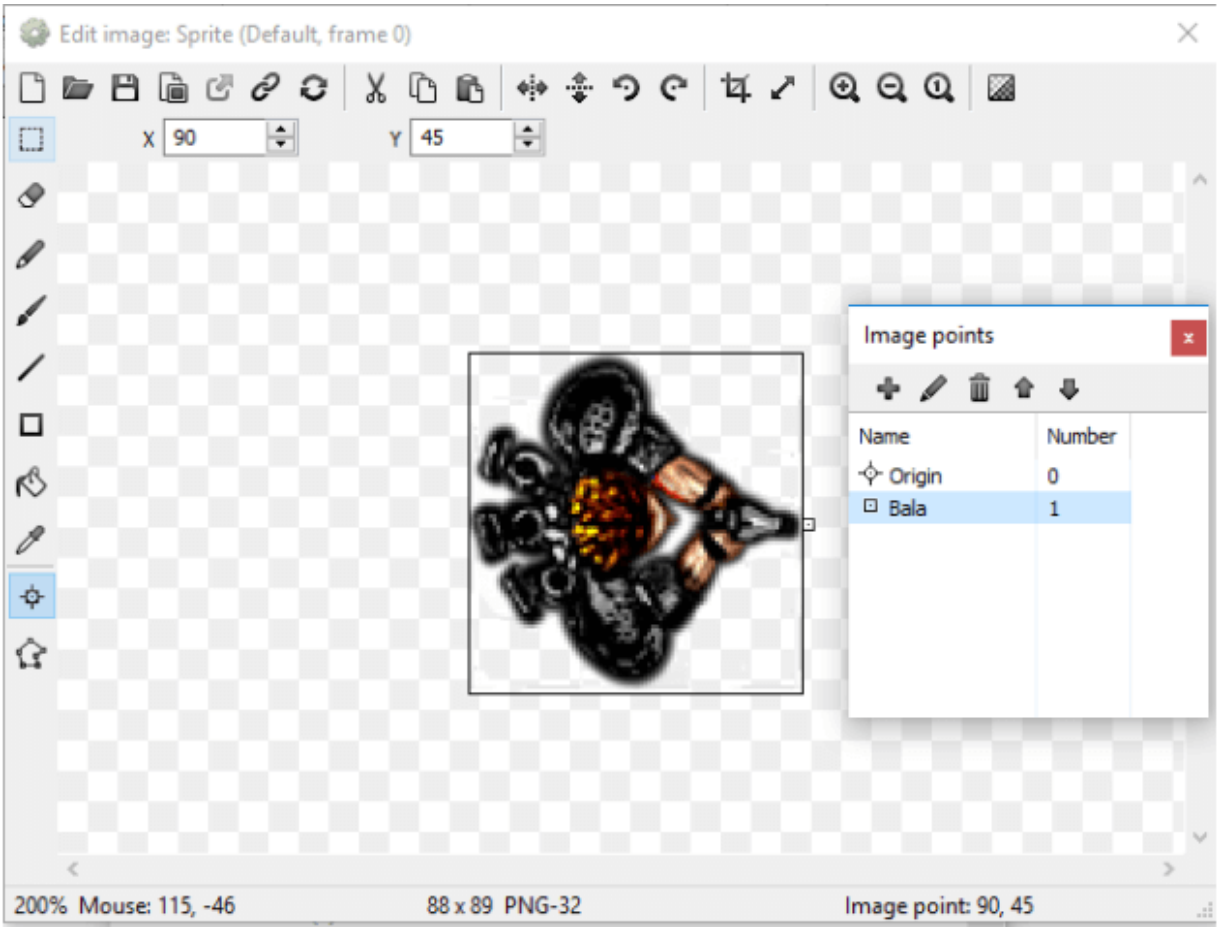


Figura 3.6: Pontos de imagem de uma arma

Não vamos criar um jogo de tiro neste livro, mas utilizaremos estes dois conceitos para criar nossas plataformas e configurar as colisões entre elas e o nosso personagem.

Colisões

Uma colisão acontece quando a área de colisão de um objeto entra em contato com a área de colisão de outro objeto. A área de colisão de um objeto pode ser definida no editor de imagem na ferramenta `Set collision polygon`. Podemos definir uma área de colisão de duas maneiras:

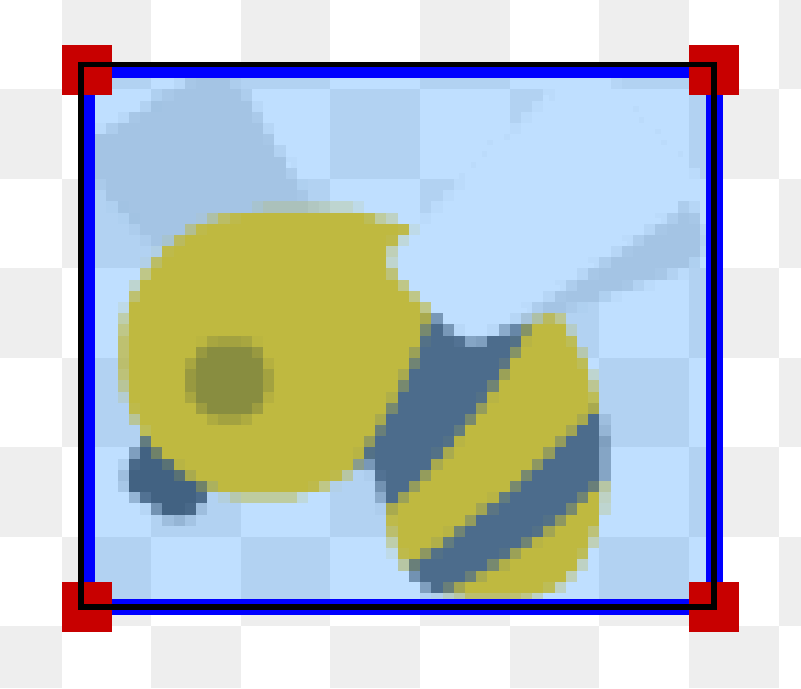


Figura 3.7: Área de colisão simples

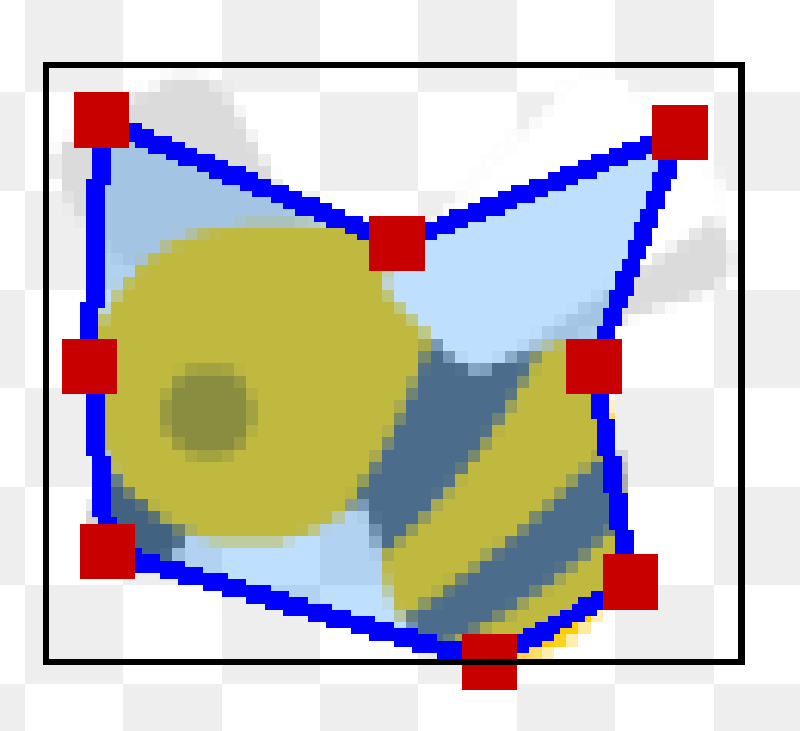


Figura 3.8: Área de colisão avançada

Na primeira imagem, temos uma área de colisão simples com somente 4 pontos, que só engloba o retângulo onde está contido o objeto. Chamamos este tipo de área de colisor em caixa. É o tipo mais comum de colisor usado nos jogos.

Para criá-lo, clique sobre a imagem com o botão direito do mouse e selecione a opção `Set to bounding box`, conforme mostra o exemplo:

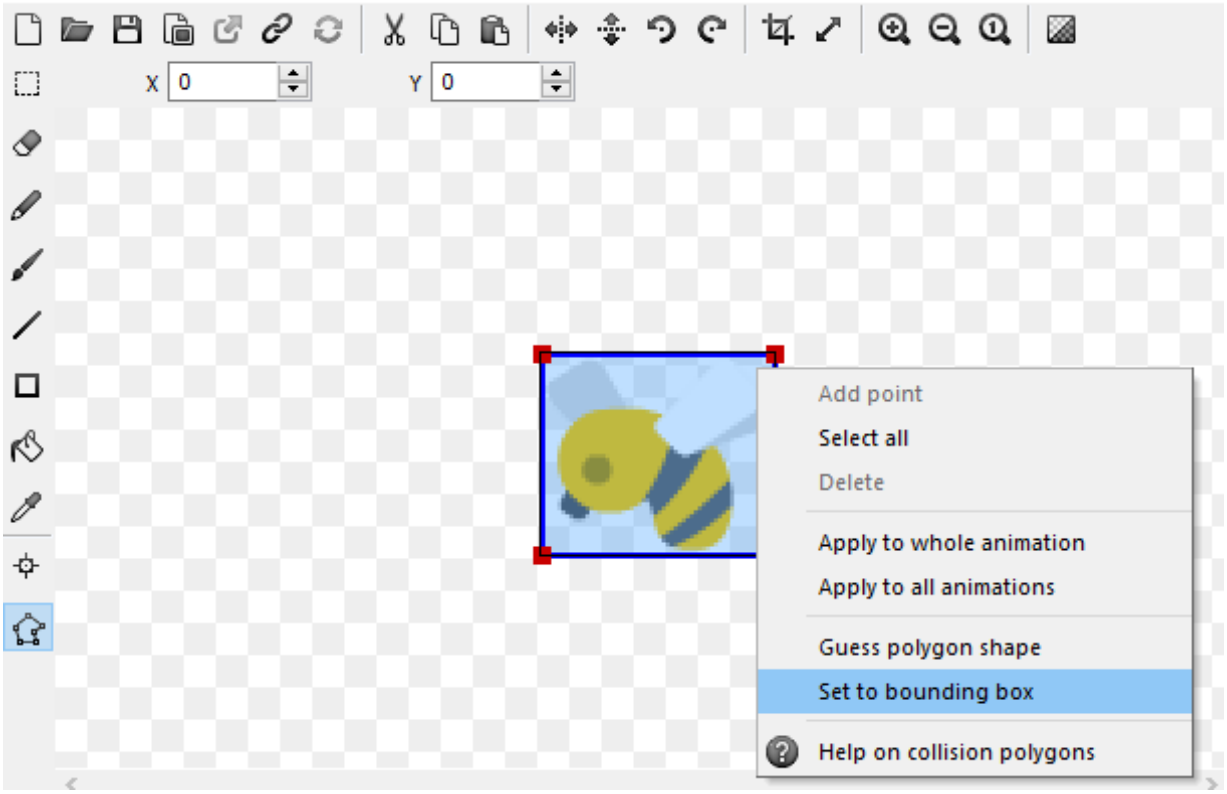


Figura 3.9: Área de colisão

A segunda imagem mostra uma área de colisão mais avançada, pois determina exatamente a forma da figura. Este tipo de colisão, por ser mais precisa, consome muito mais memória e processamento da máquina, e torna os personagens mais sensíveis a colisões.

É por isso que, em muitos jogos tridimensionais, não é difícil presenciar objetos que passam um pelo outro. Para o nosso jogo, manteremos as simples, com colisores em forma de caixa.

ÁREAS DE COLIS ES AVANÇADAS

Podemos definir uma área de colisão que contorne exatamente o nosso personagem, com o objetivo de deixar as colisões do objeto mais precisas. Para isso, basta adicionar pontos de colisões na imagem.

O Construct 2 não limita o número de pontos que podemos colocar, mas recomenda que sejam menos de oito. Para adicionar um novo ponto, basta clicar duas vezes com o botão esquerdo em cima de um dos pontos já existentes, e depois é só moldá-los como desejar.

Vale lembrar que os pontos de imagem e de origem, assim como as colisões, são aplicáveis somente para objetos do tipo `Sprite`.

3.5 Chão

Agora que temos os conhecimentos básicos sobre ponto de origem, pontos de imagem e colisões, conseguiremos construir nossos objetos. Vamos começar pelo chão. Ele será o objeto que representa o caminho onde o jogador pode pisar.

Selecione a `Layer principal` e insira outro objeto de `Tiled background`. Nele vamos importar a imagem da pasta de recursos `Chão`, chamada `chao-fase1`. Modifique o seu tamanho de forma que represente o chão por onde o jogador vai andar e coloque na posição que desejar no layout. Veja um exemplo:

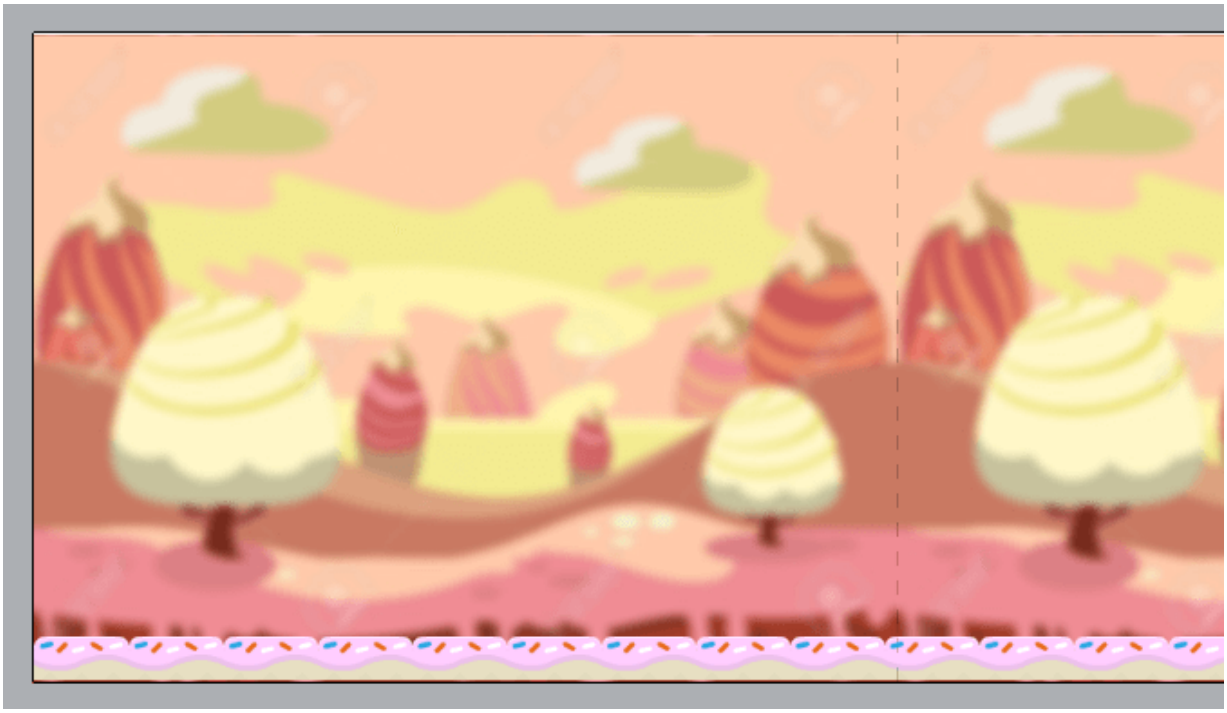


Figura 3.10: Chão da fase 1

Para que o chão não seja um fantasma que o nosso personagem vai atravessar, não se esqueça também de adicionar a `Behavior` `Solid` (sólido) nesse objeto, como fizemos no exemplo no capítulo anterior.

3.6 Plataformas

Para evitar que a nossa fase fique muito linear, vamos adicionar algumas plataformas. Elas tornam a vida do jogador mais difícil, e deixam o nosso jogo mais dinâmico e divertido. Para o projeto, vamos construir dois tipos de plataformas: móveis e estáticas.

Colocaremos todas na layer chamada `Principal` . As móveis funcionarão como elevadores para o nosso personagem, enquanto as estáticas vão servir de apoio para ele alcançar lugares altos ou que não tenham chão.

Vamos começar com uma plataforma móvel. Insira uma nova `Sprite` no layout e nomeie-a de `PlataformaMovel_fase1`. Importe novamente a imagem `chao-fase1` da pasta de recursos, colocando o seu ponto de origem embaixo no centro (*Bottom*) e a sua colisão em formato de caixa. Para isso, selecione a opção `Set to bounding box`, como já mencionado.

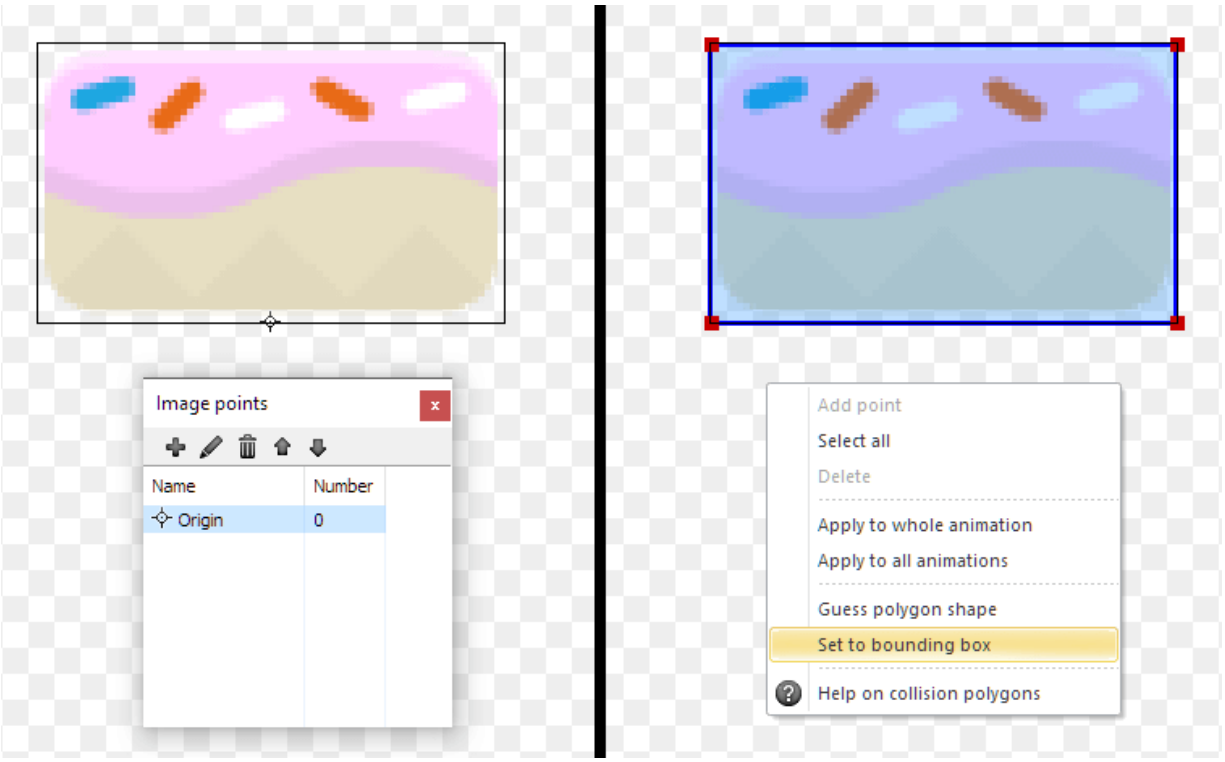


Figura 3.11: Área de colisão e ponto de origem da plataforma

Adicione a `Behavior` `Sine` (seno). Este comportamento fará com que o nosso objeto se mova de acordo com o resultado de algumas funções matemáticas. Podemos configurar para que a função atue em diversos aspectos do objeto, desde o seu movimento vertical e/ou horizontal, até sua angulação e opacidade.

Para que a nossa plataforma mova-se verticalmente, coloque o valor `Vertical` na propriedade `Movement`. Para se mover na horizontal, coloque o valor `Horizontal` na propriedade. Agora, para configurar o tempo e alcance da plataforma, vamos usar as propriedades `Period` e `Magnitude`.

O tempo é propriedade que configura a duração do movimento da plataforma para chegar de um lado para o outro (ou de cima para baixo). O alcance é a distância em pixels que a plataforma chega. No nosso jogo, recomendamos os valores a seguir:

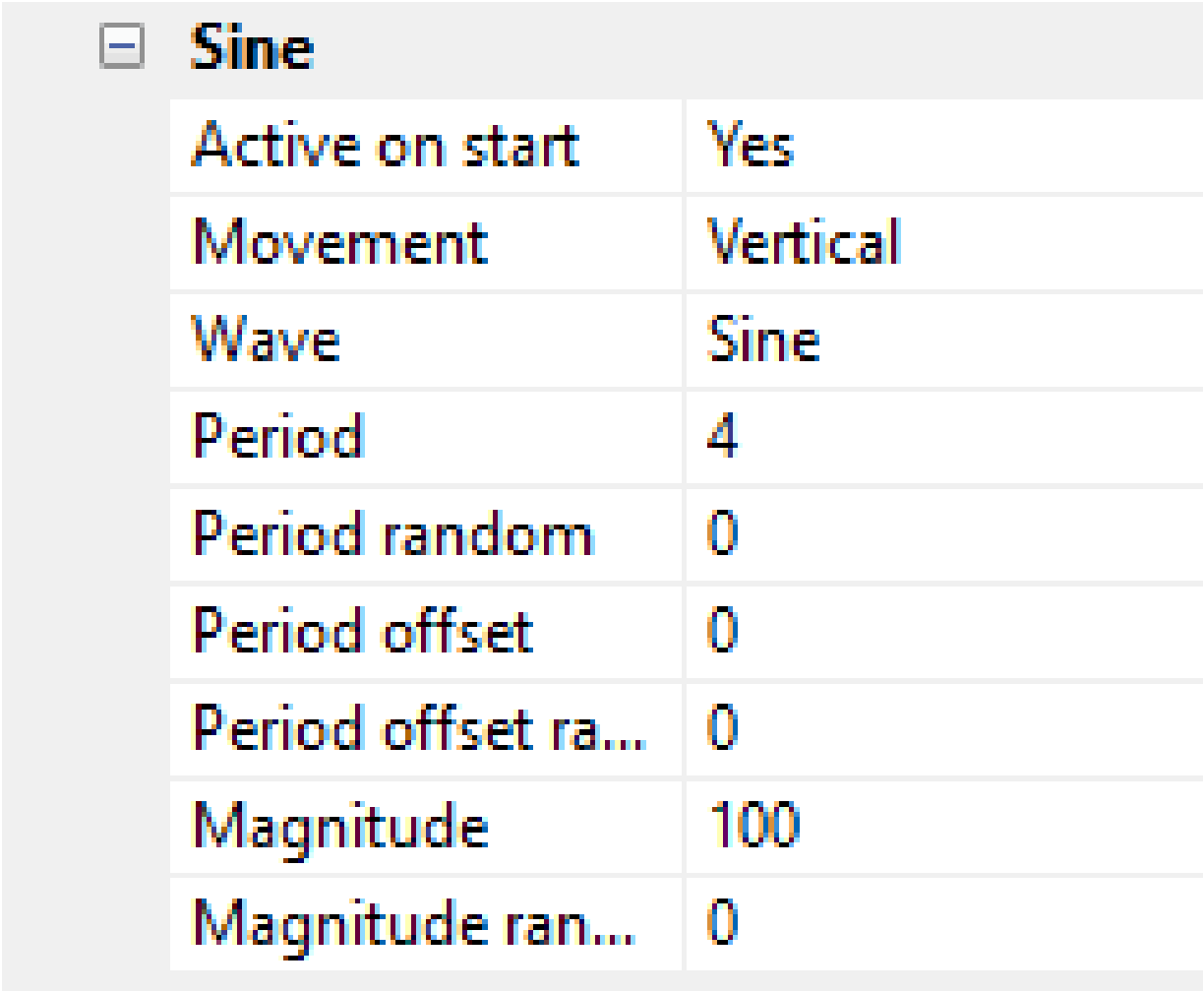


Figura 3.12: Sine Behavior

Por fim, não se esqueça de adicionar novamente o comportamento de `Solid` à plataforma; caso contrário, o personagem passará por dentro dela. Para as plataformas estáticas, apenas basta o comportamento de `Solid`. Adicione uma de cada em seu layout e faça o teste.

3.7 Decorações

As decorações nada mais são que objetos utilizados com o intuito de enfeitar e detalhar o nosso cenário. Use a sua criatividade para deixar o cenário com a sua cara. Na nossa pasta de recursos, há uma série de imagens que podem ser usadas na temática do jogo. Elas estão localizadas na pasta `Enfeites`.

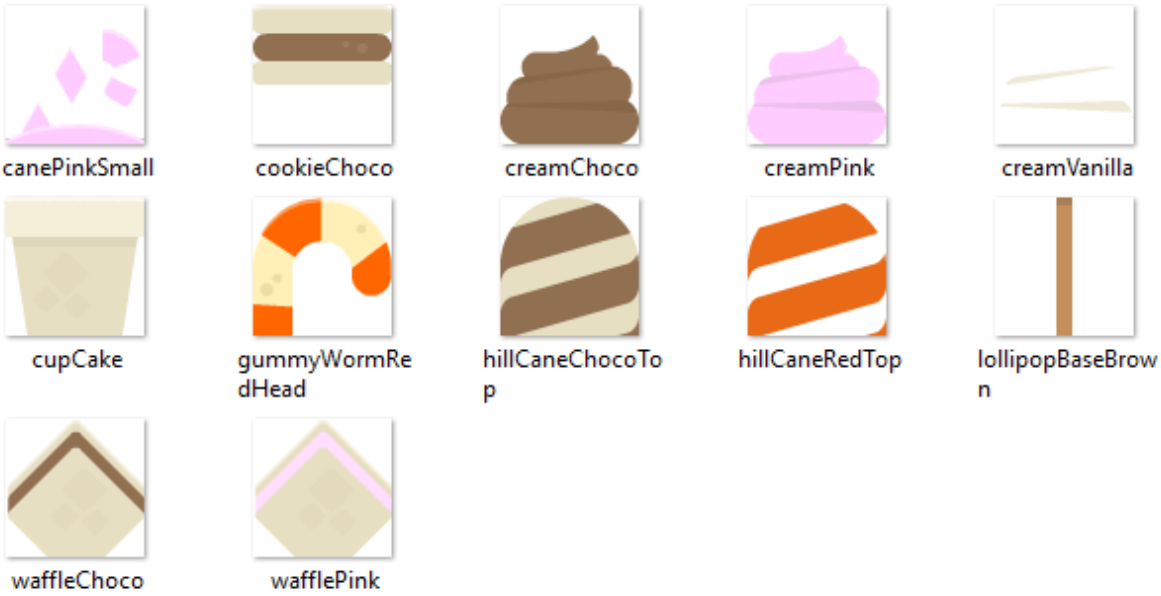


Figura 3.13: Pasta enfeites

Para adicionar uma decoração, selecione a layer chamada `Fundo` e importe as figuras como `Sprites`. Desta vez, não coloque o comportamento de `Solid`, caso contrário, o nosso personagem ficará "preso", já que estará colidindo com elas.

3.8 Barreiras invisíveis

Os layouts não possuem limitações de espaço por padrão, o que significa que qualquer objeto tem a liberdade de atravessar os seus limites verticais e horizontais. Para contornar este problema, temos

de usar paredes invisíveis. Invisíveis porque não queremos que ela seja exibida no nosso jogo, mas que apenas delimitem a área de ação dos nossos objetos.

Essa é outra técnica bastante utilizada no mercado, principalmente quando se trata de jogos de mundo aberto, como é o caso do *Grand Theft Auto*. A liberdade dada ao jogador é tanta, que é necessário criar barreiras invisíveis para que ele não saia dos limites do jogo.

Felizmente, criar uma barreira é muito simples. Basta criar uma `Sprite` na layer chamada `Fundo`, com o comportamento `Solid`. Para que ela nunca seja visível, coloque na aba de propriedades do objeto a propriedade `Initial visibility` (visibilidade inicial), com o valor `Invisible` (invisível). A figura a seguir mostra essas barreiras criadas no layout.

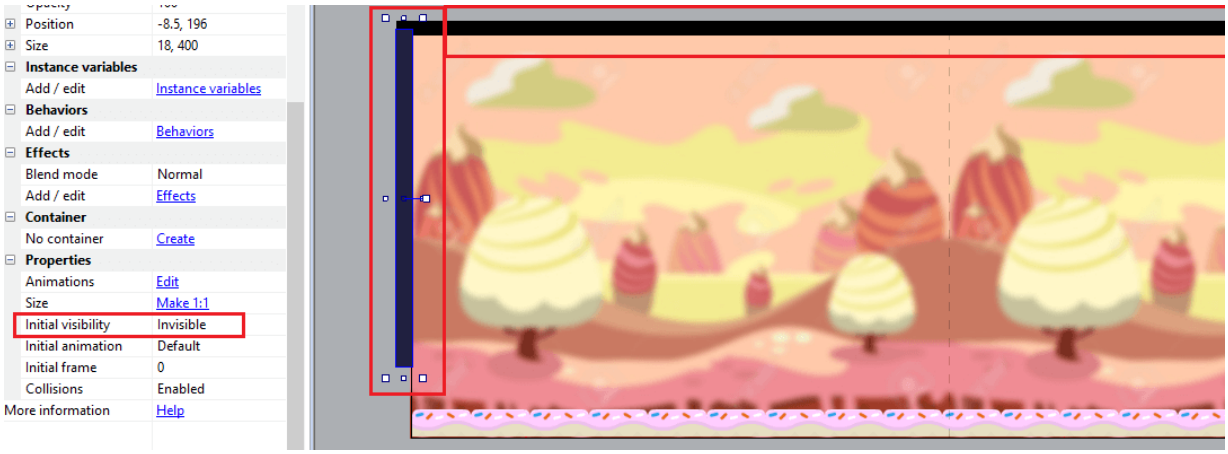


Figura 3.14: Criando barreiras invisíveis

Note que deixamos um pequeno espaço na barreira na parte inferior perto do chão. Isso servirá para que os inimigos consigam passar e "sair" do layout, e não fiquem parados no começo da fase. Não se preocupe com isso agora, vamos entender tudo no capítulo de adição de inimigos.

3.9 Adicionando a base do jogador

Agora que temos o nosso cenário pronto, vamos finalmente inserir o nosso jogador. Para evitar problemas futuros, faremos um pequeno truque aqui: em vez de já importar a figura do nosso jogador, vamos adicionar uma base retangular para ele.

Você se lembra de que as colisões podem causar problemas com o cenário? Este truque evita exatamente isso. Como vamos colocar animações para as ações do nosso personagem (pular, andar, morrer), se não utilizarmos uma base em comum, cada animação pode ter uma área de colisão diferente e causar problemas no jogo, por exemplo, ele ficar travado no cenário. Além disso, conseguiremos salvar processamento e memória, deixando o jogo mais eficiente.

Nas imagens a seguir, é possível notar a diferença de colisões entre a animação da base e do personagem que usaremos.

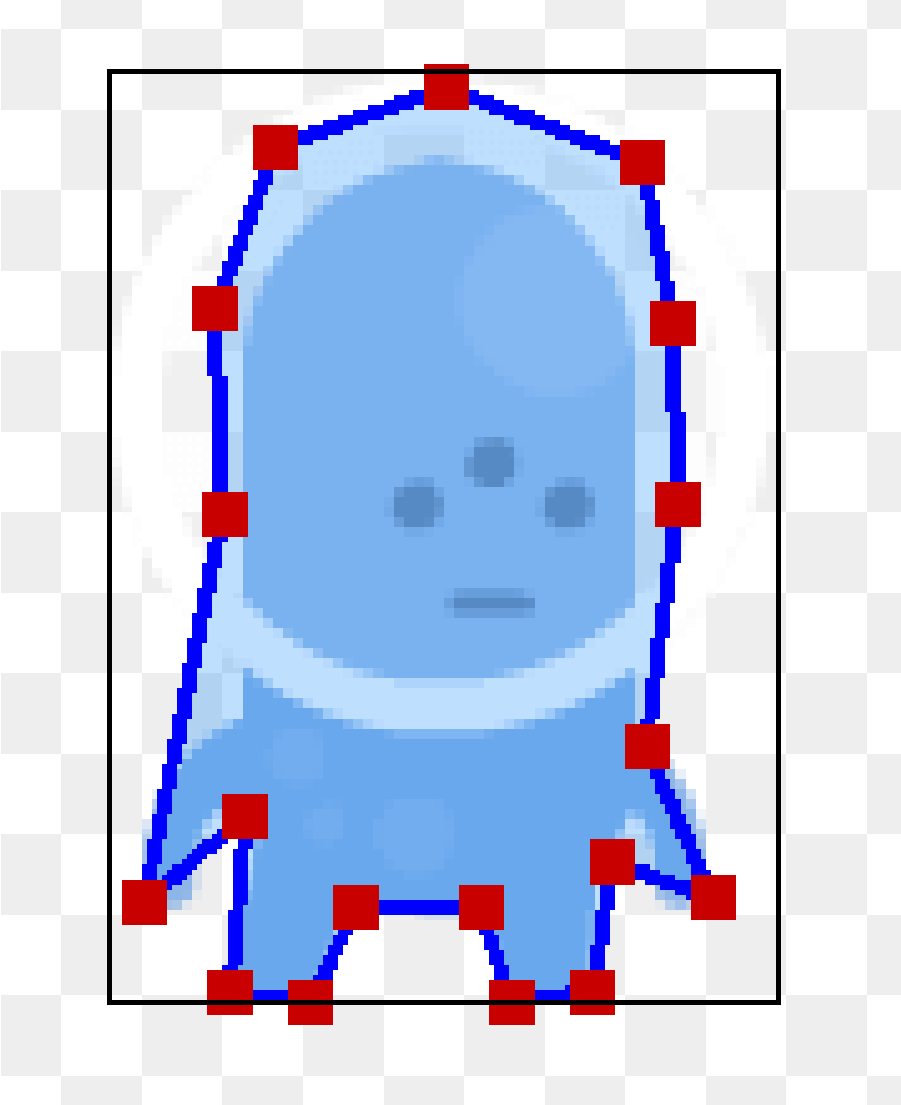


Figura 3.15: Colisões na animação do jogador

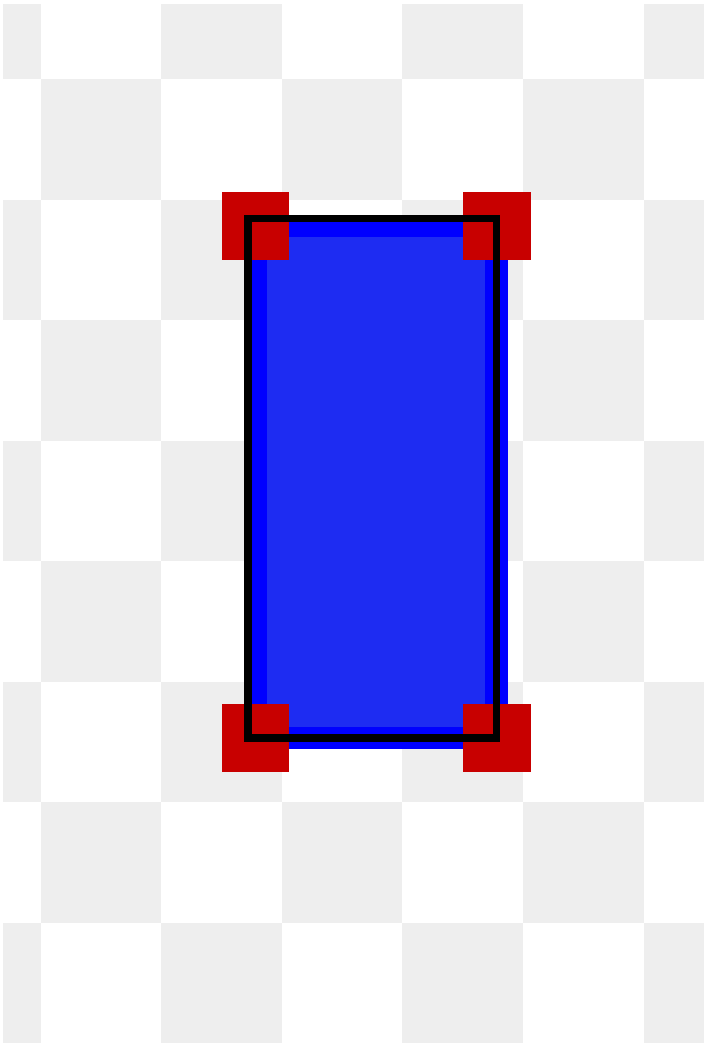


Figura 3.16: Colisões na base do jogador

Com a layer chamada `Jogador` selecionada, crie uma nova `Sprite` que será a base do jogador. Crie um retângulo nas dimensões 32x68 pixels, para que fique semelhante ao da imagem anterior. Adicione a `Behavior` `Platform` a ele.

Nas características deste comportamento, mude o valor da propriedade `Double jump` (pulo duplo) para `Enabled` (habilitado), conforme exemplifica a figura seguinte. Isso fará com que o nosso personagem ganhe a capacidade de pular duas vezes, alcançando lugares mais altos. Ao final, nomeie-o de `Jogador` .

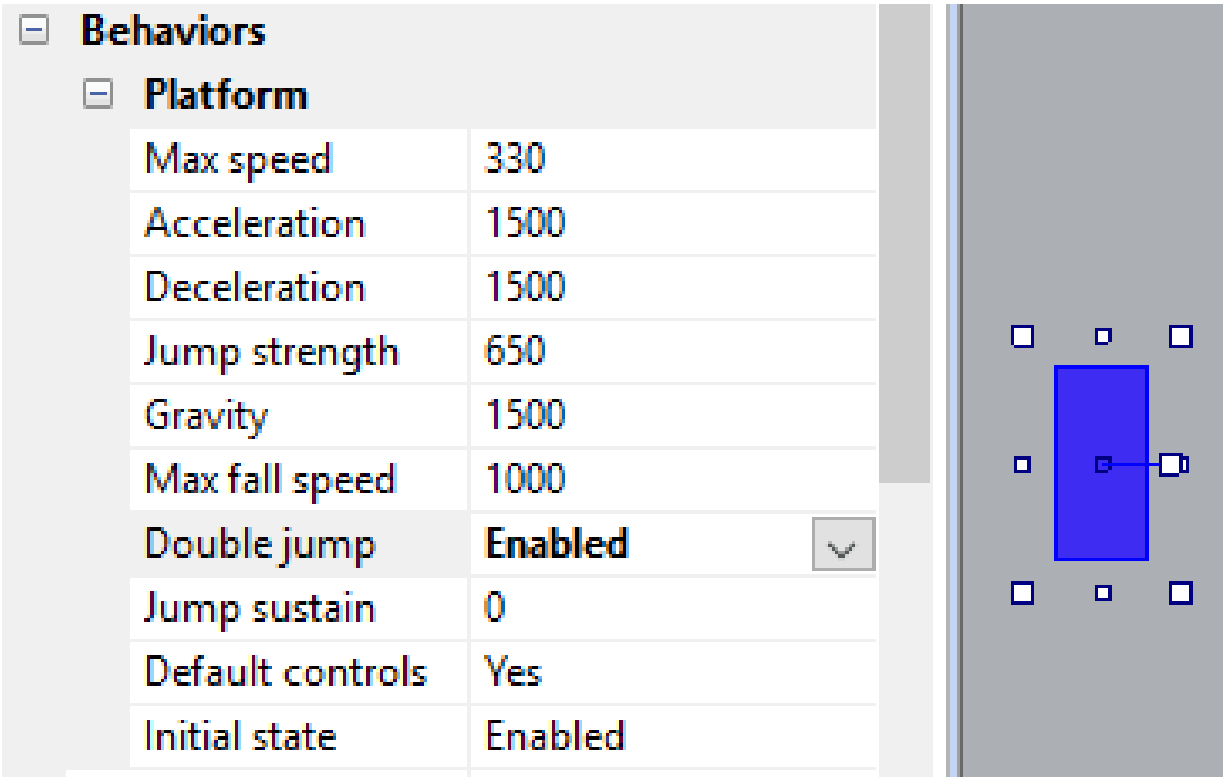


Figura 3.17: Double jump

No próximo capítulo, adicionaremos as animações do jogador e, finalmente, começaremos a programar o nosso jogo.

Capítulo 4
Movimentação e animação do personagem

Neste capítulo, vamos ver mais detalhes sobre o desenvolvimento do nosso jogo de plataforma. Aqui nós daremos vida ao nosso personagem por meio de movimentos e animações. Para realizar esta missão, veremos:

- Como funciona o editor de animações do Construct 2;
- Como usar múltiplas imagens para criar animações;
- O que são eventos e folha de eventos;
- Como utilizar os eventos para interpretar os comandos do teclado;
- Como associar eventos com ações no jogo.

Ao final deste capítulo, teremos desenvolvido toda a parte de movimentação e animação do nosso personagem, e entendido os elementos-chave para a criação de qualquer jogo de plataforma. Vamos começar?

4.1 Menus de animação

Para implementar os movimentos e as animações do nosso personagem, primeiros temos de retornar ao editor de imagens e entender como funciona os seus menus de animação. Os menus de animação são simples; na figura a seguir, destacamos seus principais elementos:

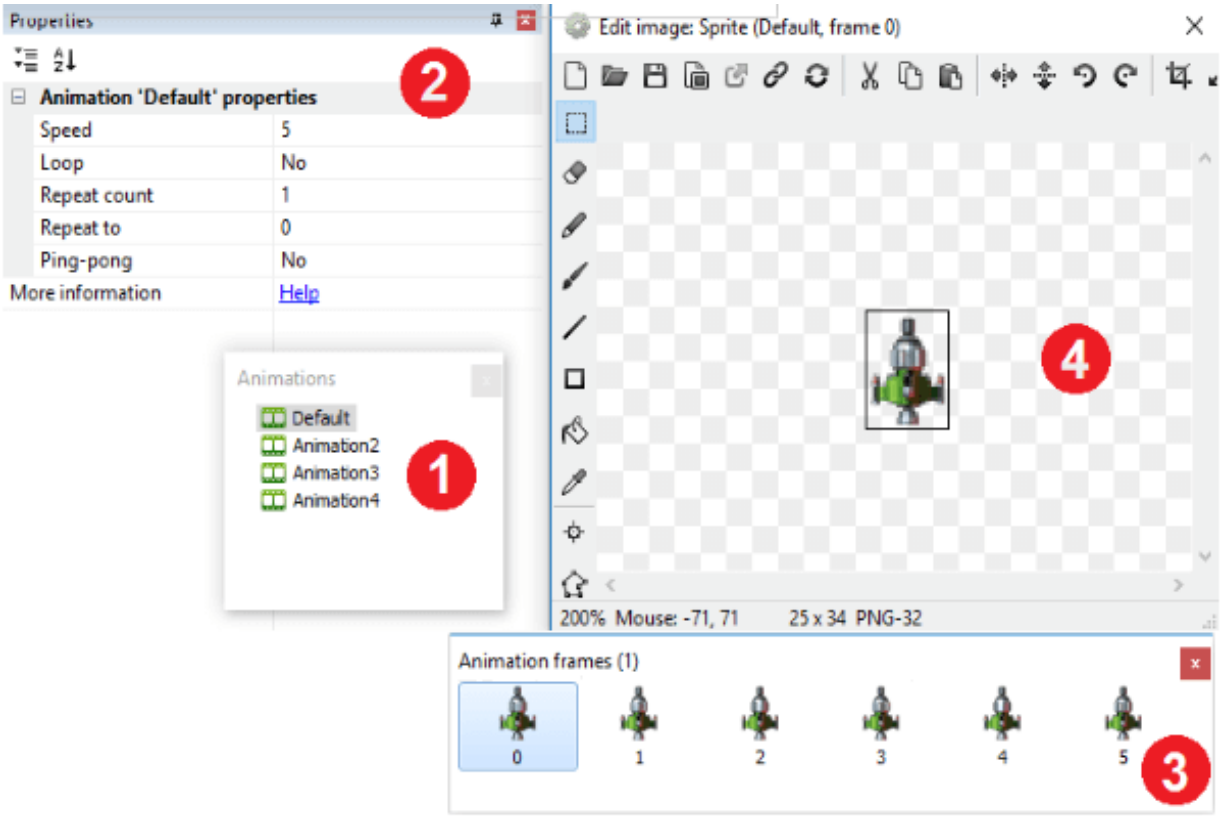


Figura 4.1: Criando animações

Cada número representa um dos elementos do menu e tem uma função específica. Vamos ver para que serve cada um deles:

1. **Barra de Animação (*Animations Bar*):** é aqui onde definimos o nome das nossas animações, e a animação `Default` já vem inserida por padrão. A não ser que modifique ou configure outra animação, ela será a usada pelo objeto associado dentro do jogo. Para cada animação criada neste menu, as configurações que veremos a seguir ficam associadas a ela. Ou seja, na prática, teremos propriedades e frames distintos para cada animação inserida aqui. Por fim, ao clicar com o botão direito sobre a animação, podemos renomeá-la, removê-la, colocá-la em pastas ou mesmo ver uma prévia de como ela está ficando. Não se preocupe que veremos tudo isso.

2. **Barra de Propriedades (*Properties bar*):** nesta barra, são definidas as características das animações. Há quatro configurações que podemos alterar:
    - Velocidade (*Speed*): a velocidade nada mais é que o número de quadros (ou *frames*) exibidos por segundo.
    - Laço (*Loop*): aqui dizemos se animação deve se repetir ao terminar ou não.
    - Número de repetições (*Repeat Count*): se a animação for configurada para se repetir, aqui podemos dizer quantas vezes – desde uma, duas até infinitas.
    - Repetir em (*Repeat to*): aqui indicamos em qual frame a animação deve voltar em caso de repetição. Na maior parte dos casos, sempre será o primeiro.
    - Ping-Pong: este efeito é para fazer um vai e volta da animação.
3. **Barra de quadros de animação (*Animation Frames*):** as animações nada mais são do que várias imagens exibidas rapidamente, dando a impressão de movimento. É nesta barra que vemos todas as imagens, ou melhor dizendo, os quadros que as formam. Aqui também podemos mudar, reordenar, adicionar ou excluir a origem dos quadros.

4. **Editor de imagem (*Edit Image*):** esta parte já vimos, é onde editamos as imagens individualmente, definimos colisões, pontos de origem, tamanho etc.

Este é o básico dos menus de animação. Agora que entendemos para que servem, vamos adicionar as animações do nosso personagem.

4.2 Animação do personagem

Vamos inserir animações para três estados do personagem: parado, andando e pulando. Chamaremos estas animações de `Parado`, `Andando` e `Pulando`, respectivamente. Com a layer `Jogador` selecionada, adicione um objeto `Sprite` e nomeie-o de `JogadorAnimacao`, como mostra a figura.

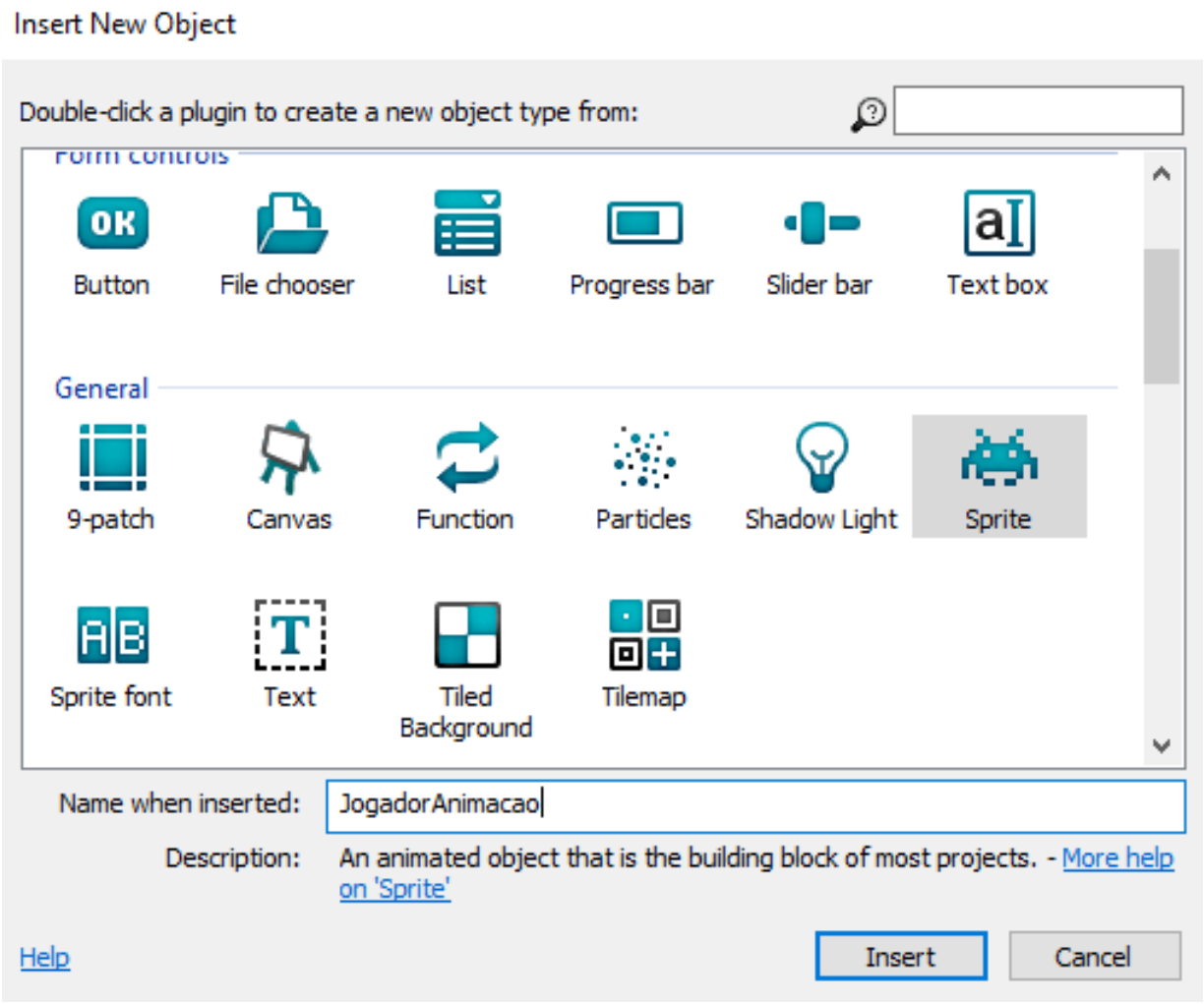


Figura 4.2: Adicionando uma nova Sprite

Em seguida, no editor de imagem, importe a imagem `Jogador-parado` que está dentro da pasta `Jogador`, na pasta de recursos que disponibilizamos. Mude o nome da animação padrão `Default` para `Parado` (lembrando que `Default` é o nome dado pelo Construct 2 por padrão à animação inicial). Ela terá somente um quadro.

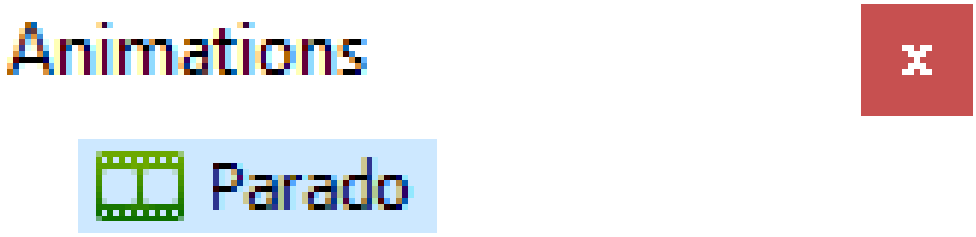


Figura 4.3: Animação Parado

Agora, crie mais uma animação na mesma `Barra de Animação`, com o nome de `Andando`.

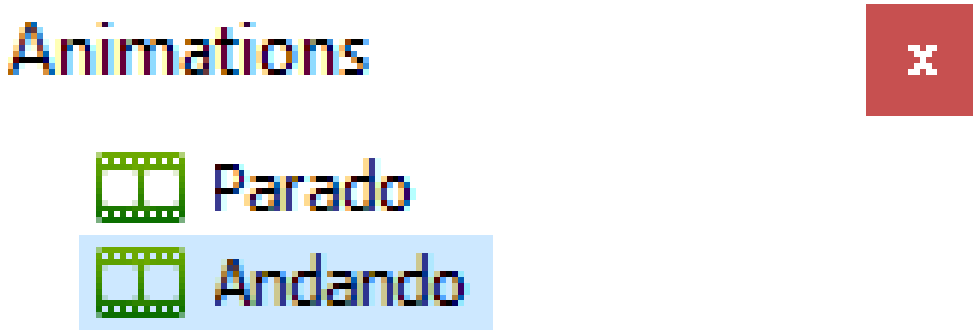


Figura 4.4: Animação Andando

Para ela, utilizaremos dois quadros e, em cada um, vamos movimentar os braços e as pernas do personagem para realmente dar a impressão de que ele está andando. As imagens que usaremos são as `Jogador-andando1` e `Jogador-andando2`, na pasta de recursos.

Temos de importar uma de cada vez. Vá até a barra de quadros de animação e clique com o botão direito do mouse na opção `Add Frame` (adicionar quadro), exatamente como mostra a figura:

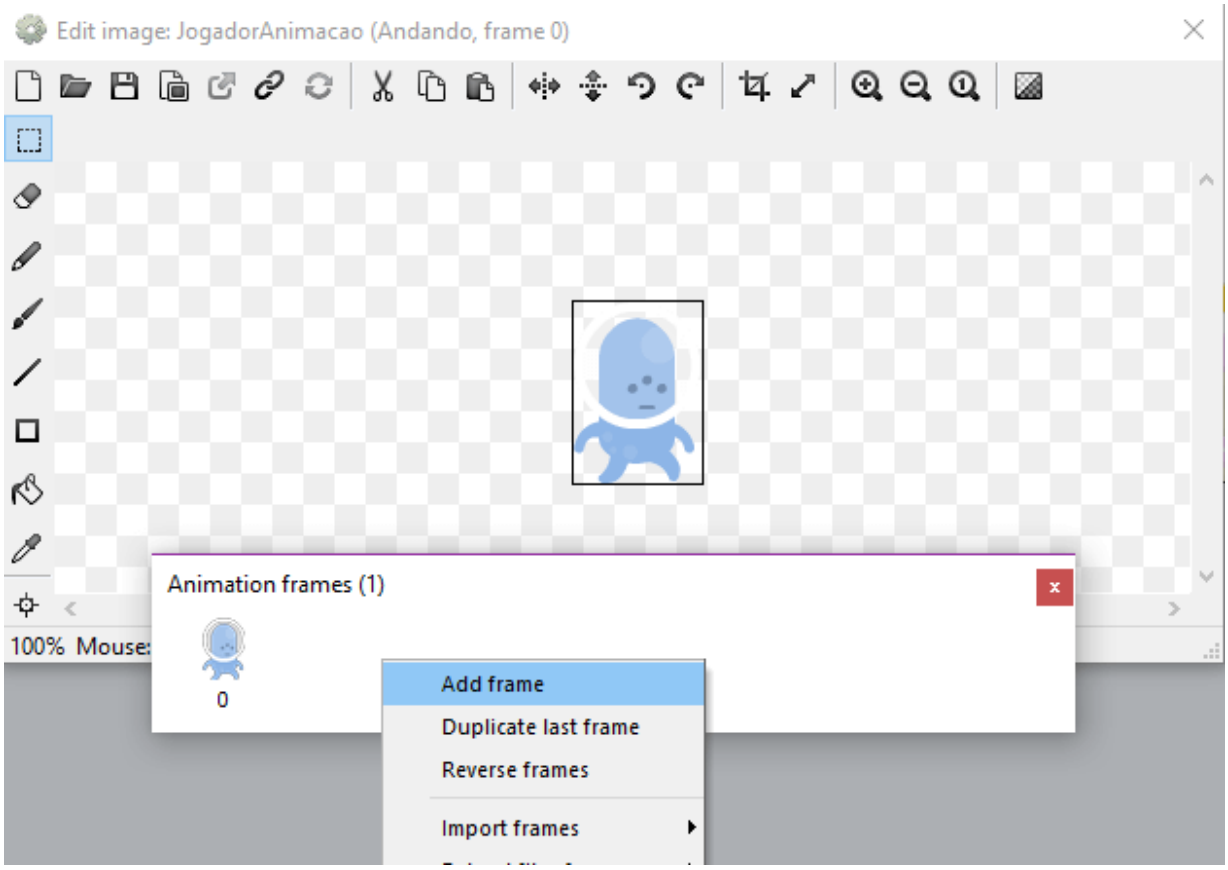


Figura 4.5: Adicionando um novo frame

Ao importar as imagens, teremos nossa animação:

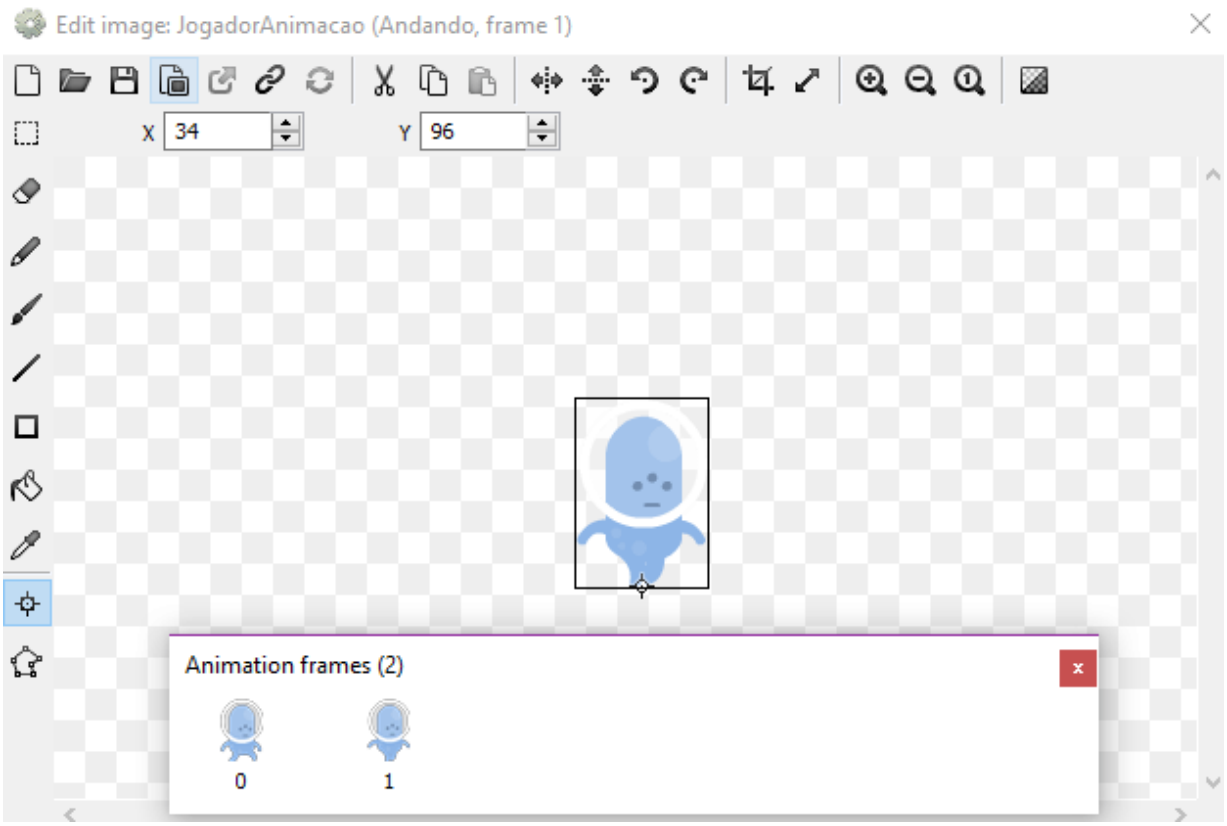


Figura 4.6: Adicionando Frames

Por fim, vamos acrescentar a terceira animação, a de pulo. Crie uma nova e nomeie-a como `Pulando`. Assim como fizemos no passo anterior, insira nela a imagem `Jogador-pulando`. Sua barra de animações deve ficar assim:

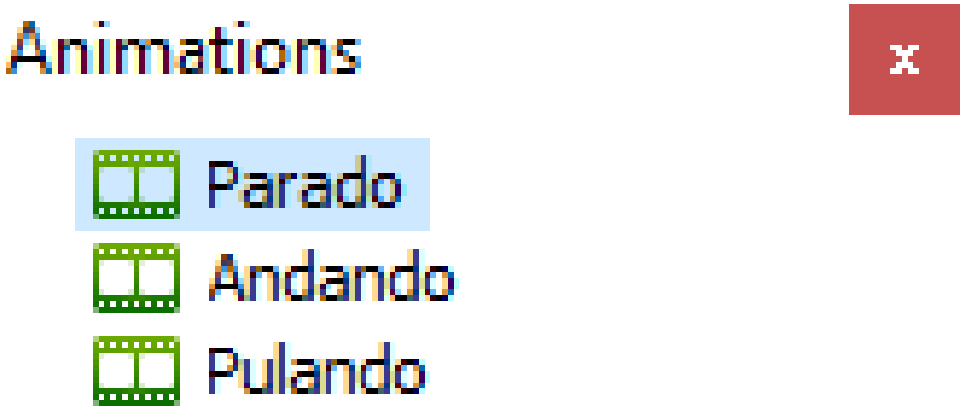


Figura 4.7: Barra de animação

Com as animações estabelecidas, não podemos esquecer de configurar corretamente o seu ponto de colisão e de origem. Lembram-se deles? Precisamos deixá-las iguais; caso contrário, o nosso personagem vai se comportar de maneira diferente a cada animação.

Vamos definir o ponto de origem embaixo no centro (*bottom*) e uma colisão simples (*set bounding box*), como mostram as duas figuras a seguir.

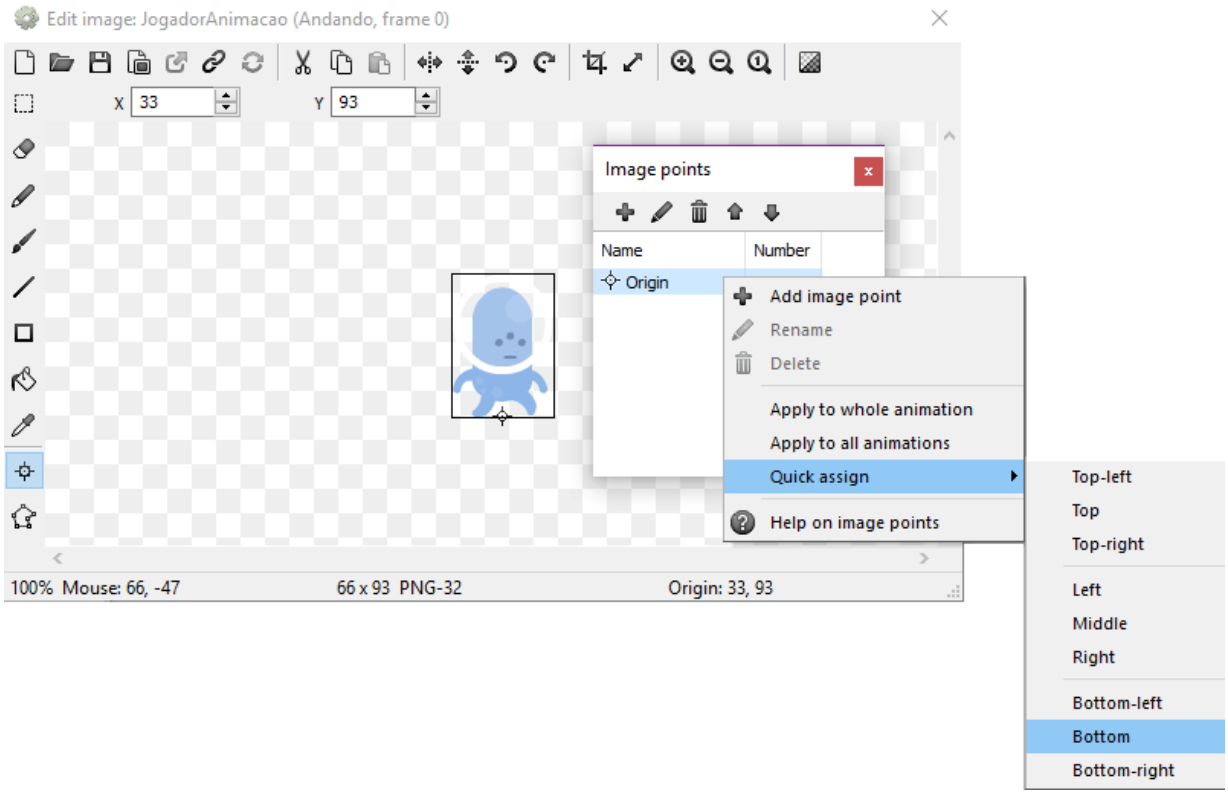


Figura 4.8: Ponto de origem embaixo no centro

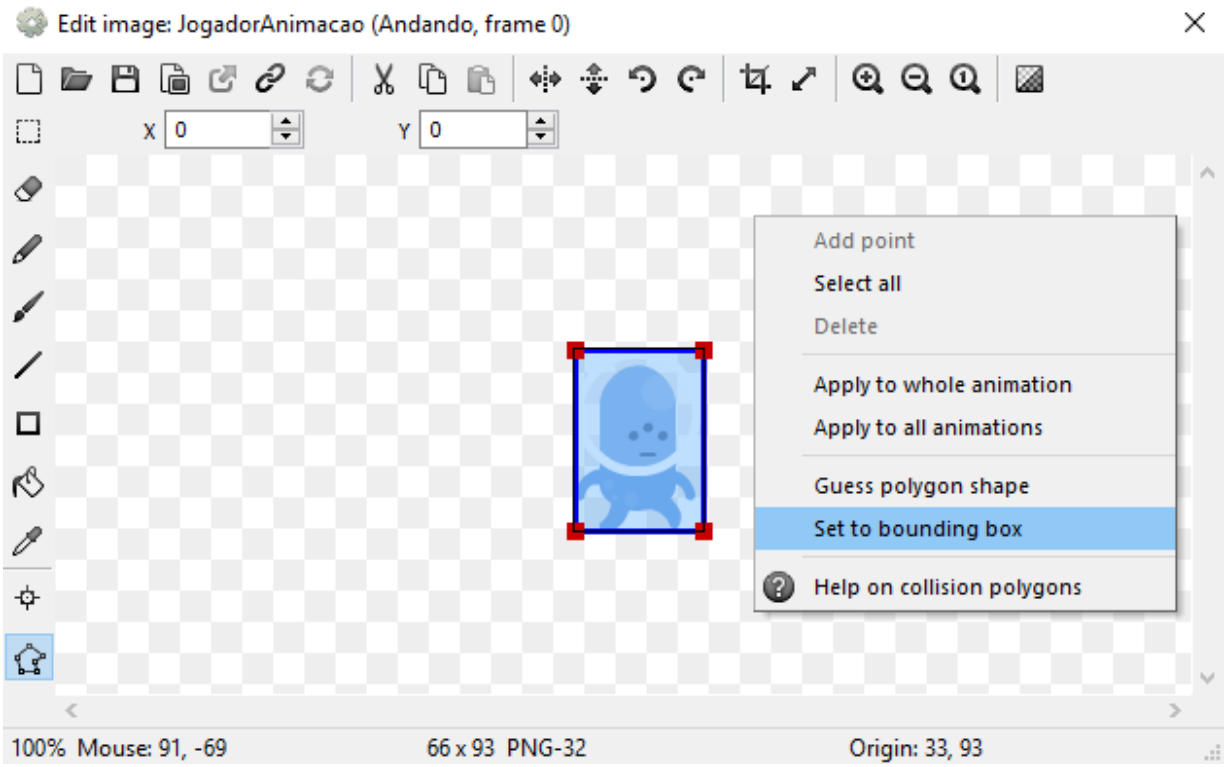


Figura 4.9: Colisão simples

Nossas animações estão perfeitas! Ficou muito legal, né? Neste ponto, fique à vontade para adicionar mais frames e detalhes a elas, e deixar o jogo com a sua cara.

Uma vez terminado, coloque a `Sprite JogadorAnimacao` em um ponto fora do layout. Por meio da folha de eventos, faremos com que ela sempre assuma a mesma posição do `Sprite Jogador`, que colocamos no capítulo anterior. Com isso, o personagem vai ser mover de forma fluida e sem problemas de colisão entre as mudanças de animação.

Entretanto, antes de colocar o nosso personagem para correr, precisamos adicionar três comportamentos a ele:

- **Pino (*Pin*):** este faz com que um objeto fique preso a outro, na mesma posição e ângulo. Precisaremos disso para que a `Sprite JogadorAnimacao` sempre fique na mesma posição da `Jogador`.

- **Piscar (*Flash*):** como o nome já indica, este comportamento faz com que o objeto pisque. Vamos usá-lo para que o nosso personagem pisque ao ser atingido por um inimigo do jogo. Isso indicará ao jogador que ele foi ferido.
- **Rolagem (*Scroll to*):** este é essencial para praticamente todos os jogos em que o tamanho do layout é sempre maior que o tamanho da janela de visibilidade do jogador. Isso porque este comportamento configura a câmera do jogo para acompanhar o movimento de um objeto. Sem ele, perderíamos o personagem de vista toda vez que ele ultrapassasse os limites da tela. Em nosso jogo, faremos com que a tela sempre siga o nosso personagem.

Selecione a `Sprite JogadorAnimacao` e insira esses três comportamentos nele. Lembre-se de que, para adicionar um comportamento, basta clicar no botão `Add behaviour` na `Barra de propriedades` do objeto. Ao final deste passo, ele deverá estar assim:

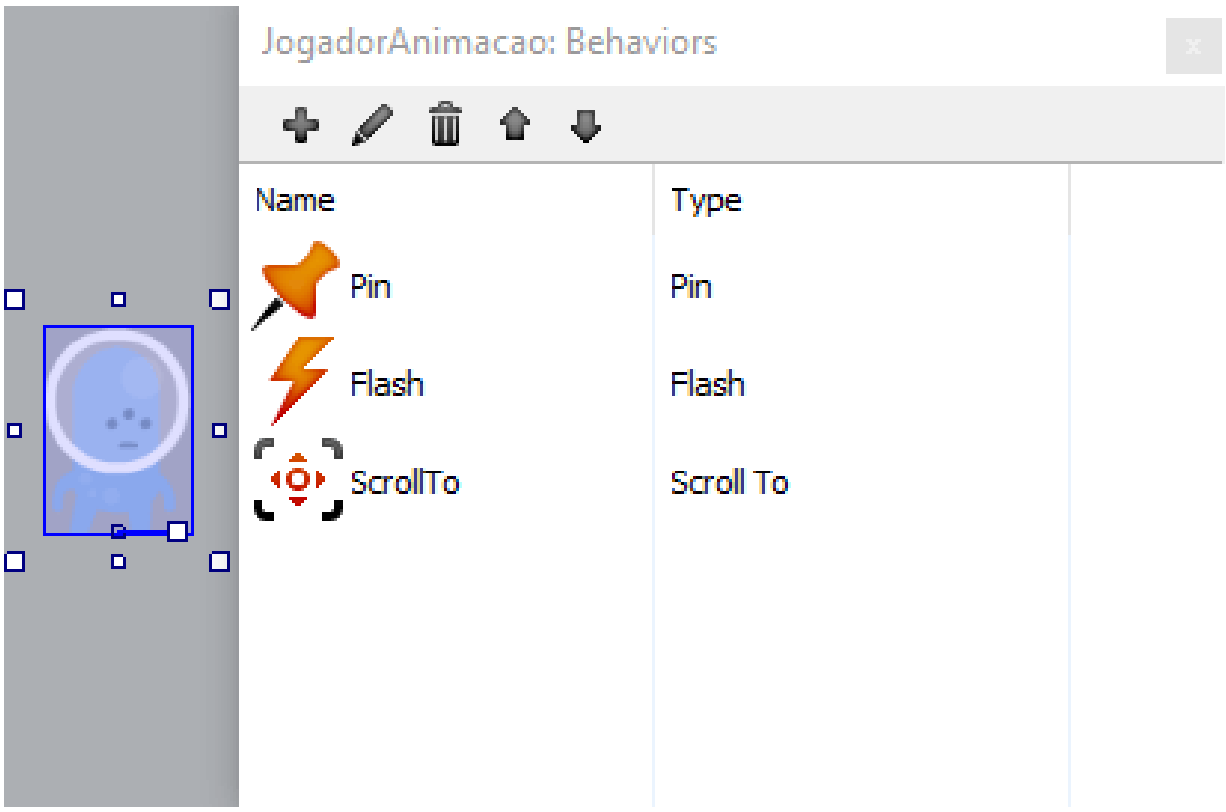


Figura 4.10: Adicionando as Behaviors do jogador

Nosso personagem está quase pronto! O que precisamos para deixá-lo perfeito é adicionar uma maneira de movimentá-lo. No Construct 2, temos vários jeitos de fazer um objeto se movimentar. As maneiras que usaremos neste livro são:

- Mouse;
- Teclado (*Keyboard*);
- Toque (*Touch*).

Na folha de eventos, temos como configurar as animações e as movimentações de acordo com o que for pressionado pelo jogador, seja por meio de uma tecla do teclado, de um clique do mouse ou um toque na superfície (para dispositivos como tablets e smartphones). Na próxima seção, veremos como fazer isso.

4.3 O que são eventos e folhas de eventos?

Como já mencionado, nas folhas de eventos ficam contidas toda a "programação" do jogo criado no Construct 2. Utilizamos aspas em programação, porque não escrevemos sequer uma linha de código; tudo é feito com eventos e blocos lógicos.

Os eventos são métodos pré-construídos pela plataforma e possuem funções bem específicas dentro do jogo. Cada evento é constituído por condições e ações, como mostra a figura:

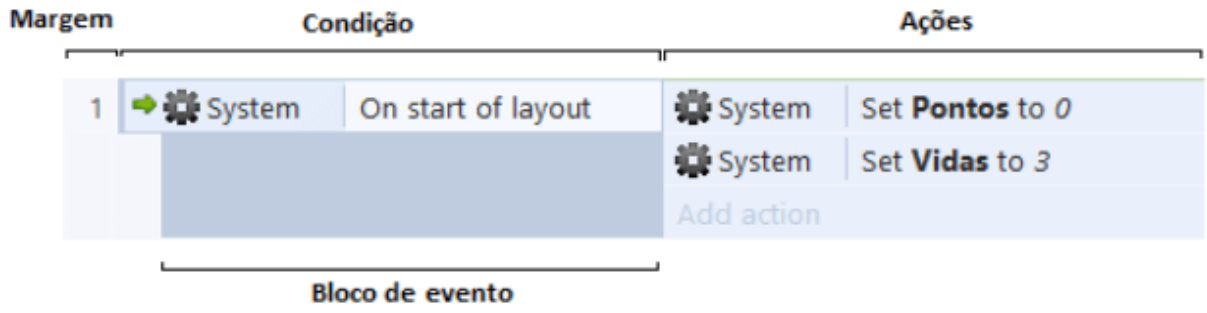


Figura 4.11: Estrutura dos eventos

Cada condição dispara um ou mais eventos. Eles nos fornecem as ferramentas básicas que precisamos para criar desde jogos simples aos mais sofisticados. Alguns exemplos de eventos que usamos no dia a dia são:

- Colisão entre objetos;
- Toque ou clique na tela;
- Tela pressionada no teclado e/ou *joystick*;
- Início/fim do carregamento de um layout;
- E a lista é longa.

O Construct 2 executa todas as linhas de eventos que estiverem contidas na folha de eventos em cada instante, que ele chama de *tick*. O *tick* (ou instante) é uma unidade de tempo que executa os eventos da folha e, em seguida, redesenha a tela.

Geralmente, os monitores atualizam o display sessenta vezes por segundo e, para que as animações do jogo fiquem as mais suaves possíveis, o Construct 2 tenta igualar esta atualização. Isso significa que a folha de eventos também é geralmente executada igualmente sessenta vezes por segundo, redesenhando a tela a cada instante. Além disso, as linhas de eventos são executadas de cima para baixo, então, os eventos no topo da folha são feitos primeiro.

Para cada condição atendida de um evento, o Construct 2 dispara um ou mais eventos. As condições validam se os critérios de ativação de um evento foram executados. Se forem verdadeiros, as ações dos eventos são executadas; caso contrário, não.

Veja um exemplo na figura anterior. Para a condição `On Start of layout` (ao iniciar o layout), as ações das variáveis `Pontos` e `Vidas` são setadas para zero e três, respectivamente. Bem bacana, né?

O Construct 2 nos oferece uma grande variedade de objetos para utilizar no jogo, e cada um contém sua própria lista de condições e ações com que podemos interagir na folha de eventos. Por exemplo, se adicionarmos um objeto de áudio, será possível interagirmos com as ações de áudio, como tocar, pausar, alterar volume etc.

Porém, independente de quantos objetos coloquemos em cena, o Construct 2 oferece o `System Object` (objeto de sistema). Neste objeto especial, tratamos todos os eventos que independem dos objetos, por exemplo: laços de repetição, salvar e carregar recursos, comparar valores etc.:

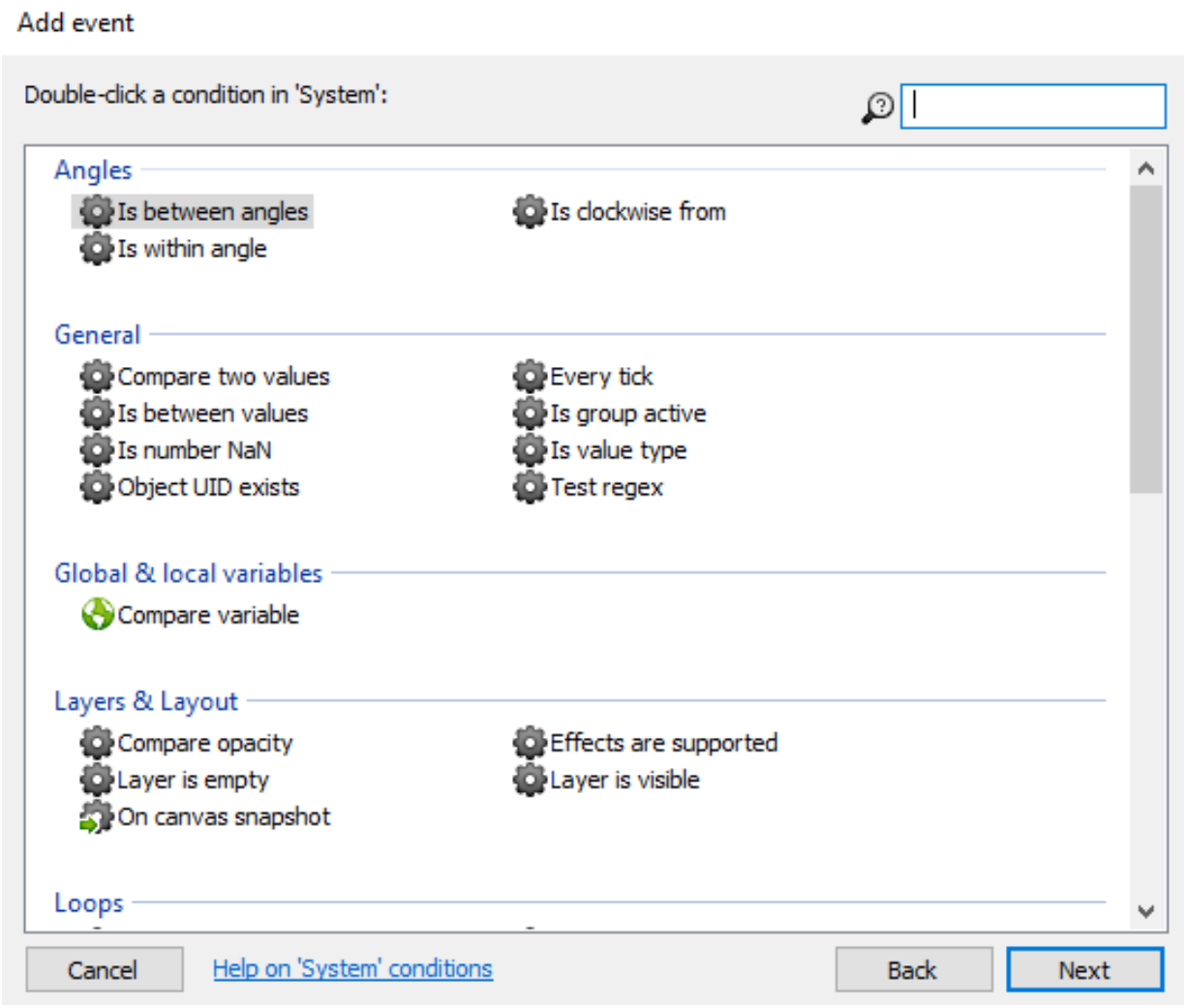


Figura 4.12: Opções do objeto Sistema

Não se preocupe que utilizaremos bastante este objeto no nosso jogo. Além disso, para fins de organização, existem outros dois recursos essenciais na folha de eventos: os grupos e os comentários.

Os grupos são blocos que têm a função principal de organizar os eventos. Temos a liberdade de criar estes grupos para colocar eventos de um mesmo contexto juntos, melhorando a legibilidade das folhas que podem ficar extensas com o desenvolvimento dos jogos.

Basta apenas arrastar os eventos para dentro desses grupos. Eles podem ser recolhidos e expandidos usando o botão, e sempre são dotados de um nome e uma possível descrição. Neste projeto, teremos os eventos separados em cinco grupos distintos: **Sistema**, **Jogador**, **Inimigos**, **Itens coletáveis** e **Botões**.

Figura 4.13: Grupos

Já os comentários são como notas de fundo amarelas, ficando acima do evento que descrevem, para o usuário lembrar do que se trata o evento ou o bloco. Também é bem útil para folhas muito extensas.

Para adicionar um comentário, clique com o botão direito do mouse em um evento, ou em um espaço vazio na folha de eventos, e escolha a opção `Add comment` (adicionar comentário). Você também pode selecionar o evento desejado e clicar no atalho de teclado `Q`. A figura a seguir mostra dois eventos com uma linha de comentário acima.

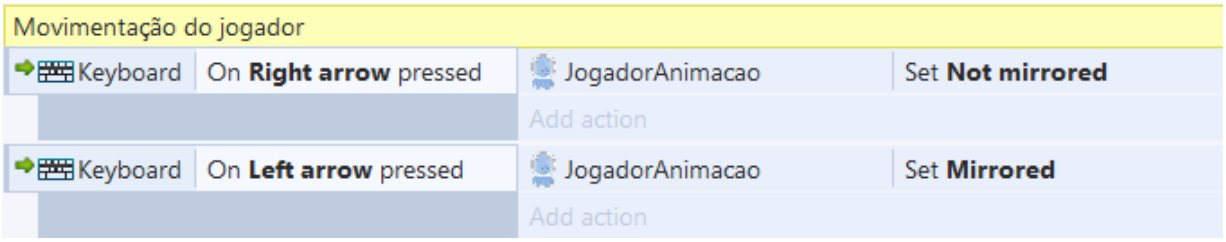


Figura 4.14: Comentários

Agora que já temos uma melhor noção de como funcionam a folha de eventos e os eventos em si, vamos começar a programá-los para movimentar o personagem.

4.4 Movimentação do personagem por eventos

Já preparamos todo o terreno para que o nosso personagem se movimente, falta somente adicionarmos a cereja em cima do bolo. Abra a página `Event sheet 1` que está dentro da pasta `Event Sheets`. Nessa folha, vamos adicionar um novo evento clicando em `Add event`, na parte superior da tela:

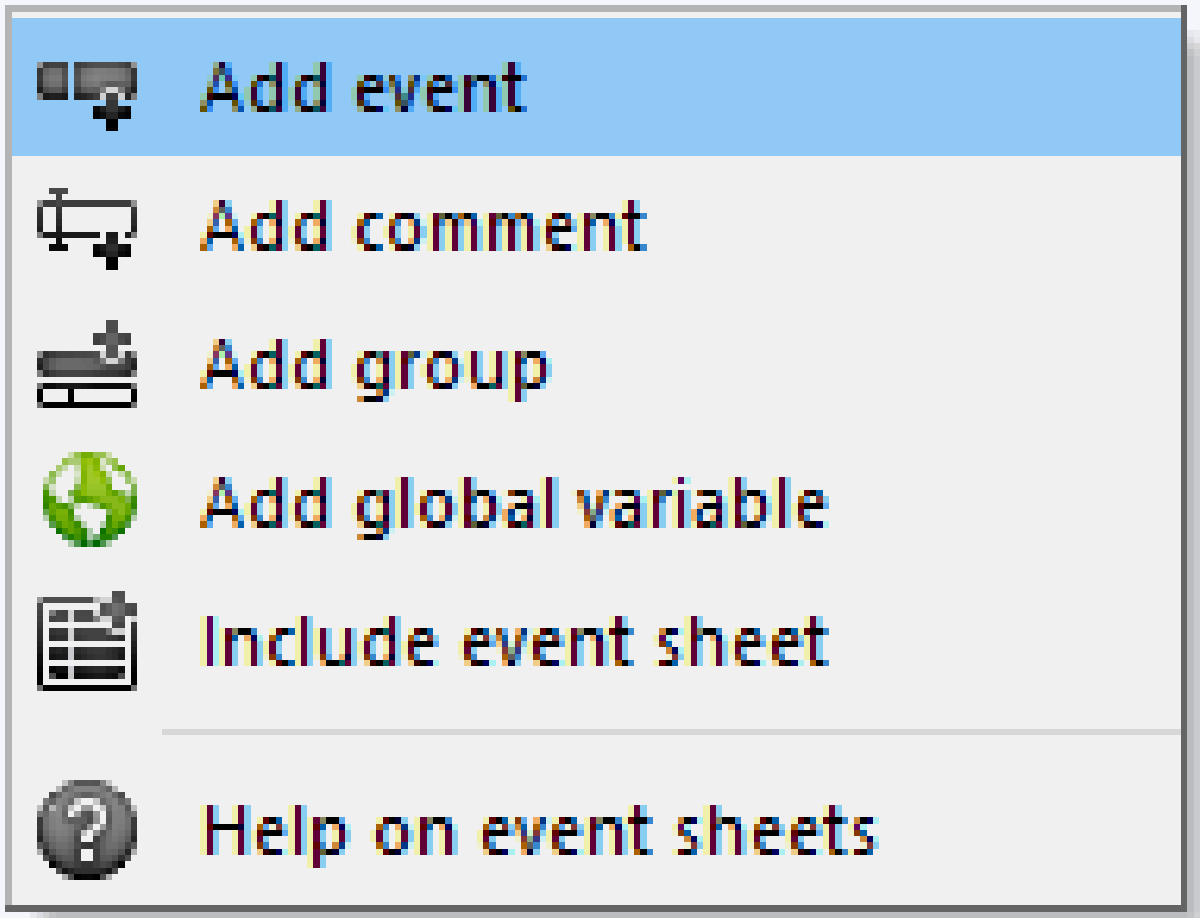


Figura 4.15: Add event

Na janela que abrir, vamos ver todos os objetos que inserimos até agora no nosso jogo. Além deles, também veremos o `System object`, o objeto que citamos que está sempre disponível, independente da quantidade de objetos do seu layout.

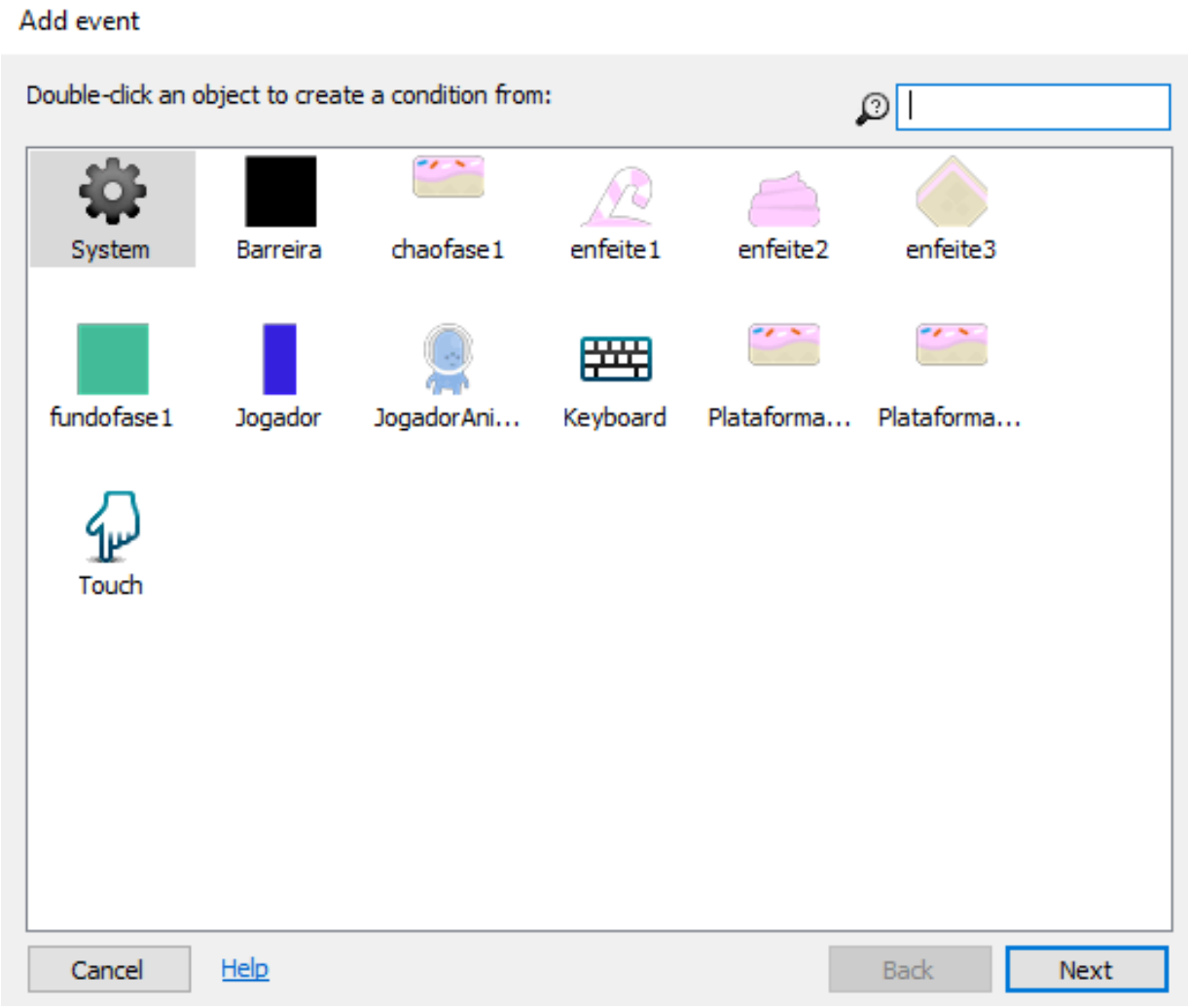


Figura 4.16: Objetos importados

Muito bem. Neste evento que acabamos de criar, configuraremos os movimentos de animação do nosso personagem para que correspondam às setas do teclado, desta maneira:

- **Tecla para cima (?):** animação de pulo;
- **Tecla para direita (?):** animação de movimentação para a direita;
- **Tecla para esquerda (?):** animação de movimentação para a esquerda.

Para começar, dê um duplo clique no objeto `Keyboard` (teclado) que importamos no projeto e clique em `Next` . Em seguida, surgirão todas

as condições associadas ao objeto de teclado. Vamos escolher a opção `On key pressed` (tecla pressionada). Esta opção checa se alguma tecla foi pressionada pelo usuário:

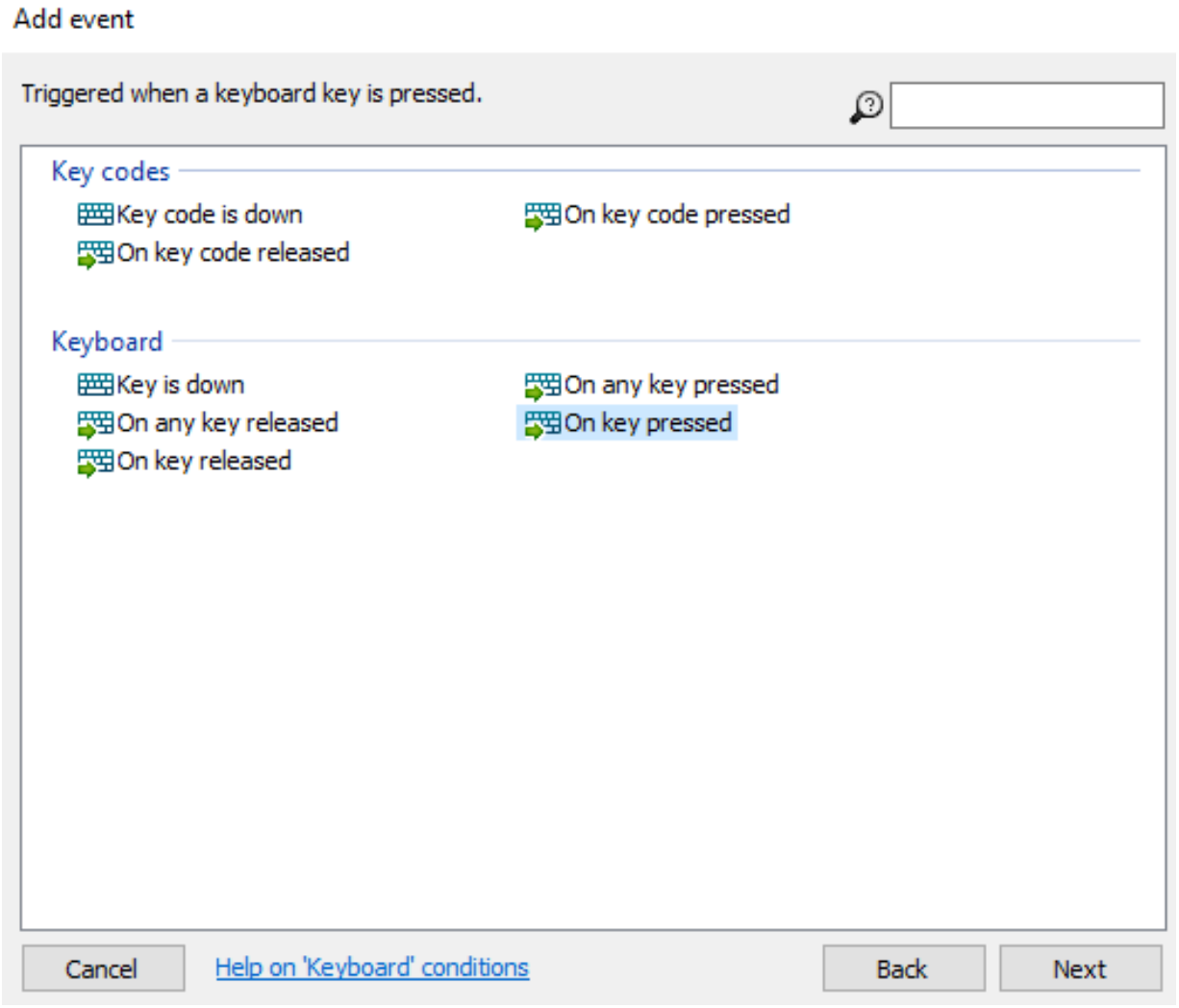


Figura 4.17: Condições do objeto Keyboard

Agora vamos escolher qual tecla deverá ser pressionada para a condição ser válida. Para isso, na próxima aba `Parameters for keyboard: On key pressed` (parâmetros para o teclado: tecla pressionada), clique em `click to choose` (clique para escolher). Na aba `Choose a key` (escolha uma tecla), selecione a seta do seu teclado para a direita (`Right arrow`), e clique em `OK` ; isso fará como

que a condição seja clicar com a seta direita do teclado, como mostra a seguir:

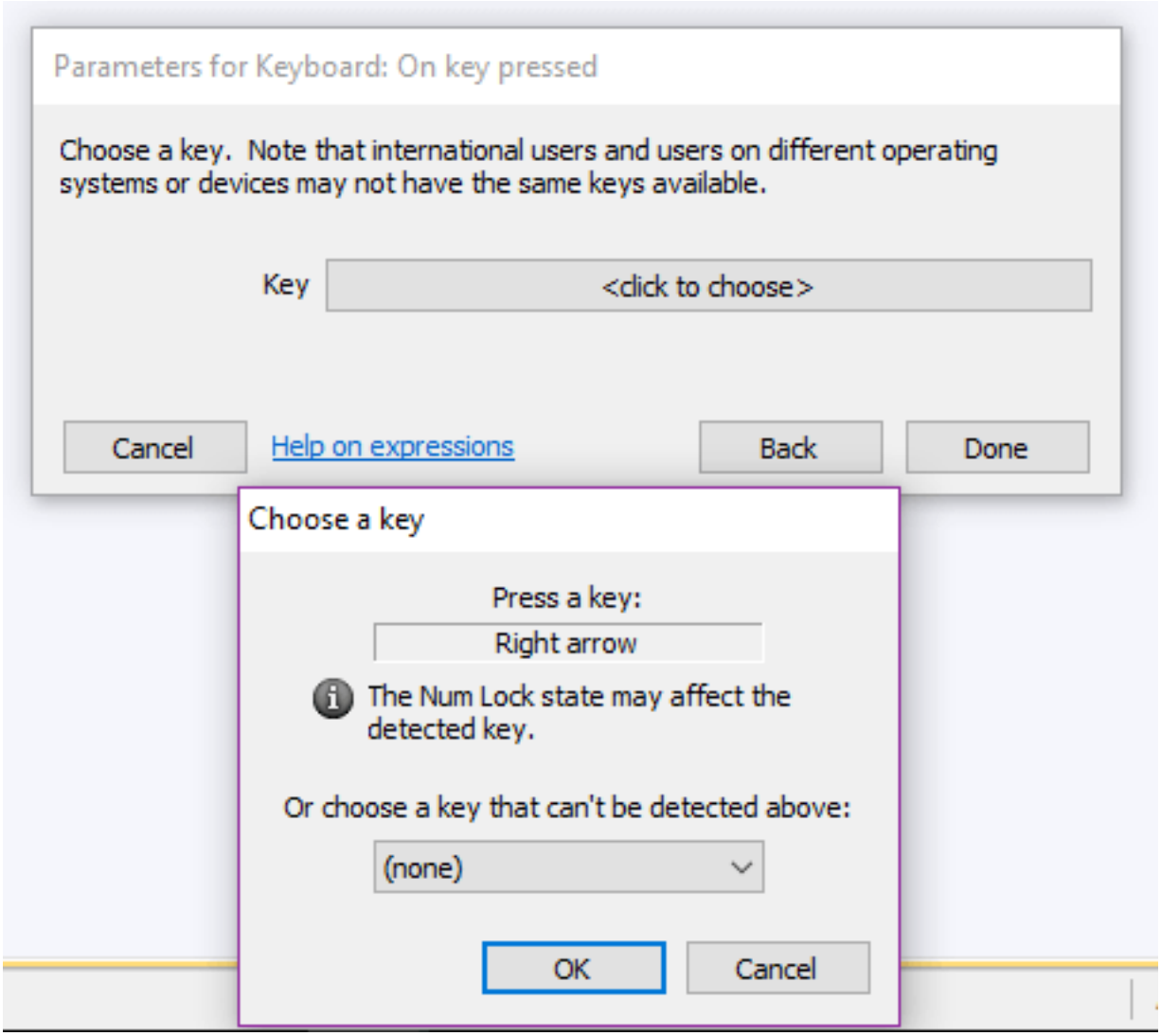

Parameters for Keyboard: On key pressed
Choose a key. Note that international users and users on different operating systems or devices may not have the same keys available.
Key
<click to choose>
Cancel
Help on expressions
Back
Done
Choose a key
Press a key:
Right arrow
The Num Lock state may affect the detected key.
Or choose a key that can't be detected above:
(none)
OK
Cancel

Figura 4.18: Escolhendo a tecla

Depois, volte para a aba `Parameters for keyboard: On key pressed`. Verifique se a tecla `Right arrow` foi mesmo selecionada e clique em `Done`:

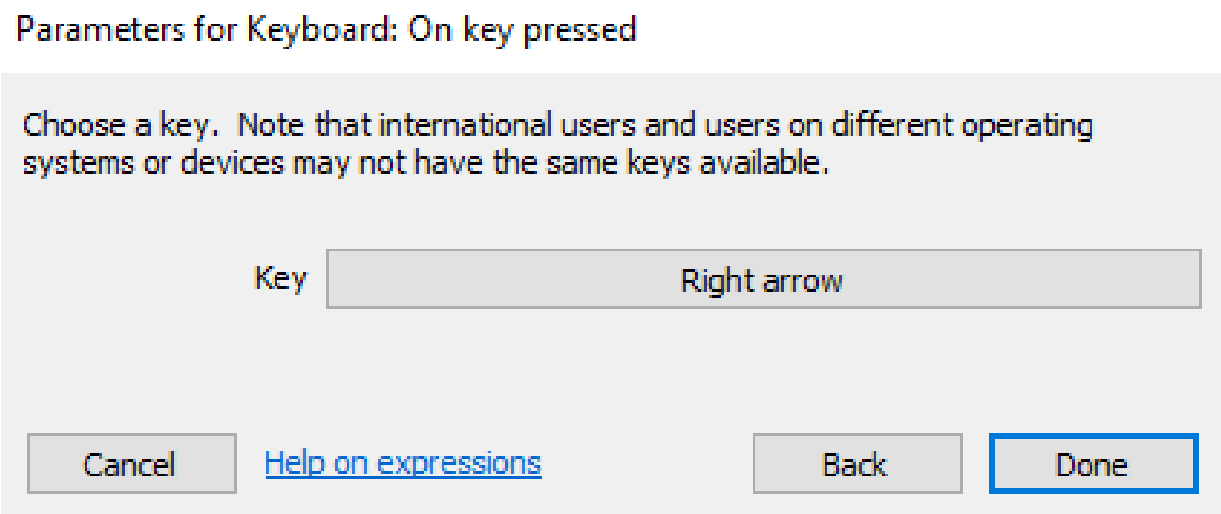

Parameters for Keyboard: On key pressed
Choose a key. Note that international users and users on different operating systems or devices may not have the same keys available.
Key
Right arrow
Cancel
Help on expressions
Back
Done

Figura 4.19: Parâmetros da tecla

Agora que estamos verificando a condição da tecla direita do teclado ser pressionada, vamos adicionar uma ação. Esta será trocar a animação de parado para andando, tomando o cuidado para que ele esteja sempre virado para a direção que o jogador estiver indo.

Para fazer isso, usaremos a ação `Mirror` (espelho). Com ela, podemos dizer ao Construct 2 para que ele reflita a imagem quando necessário. Por isso, vá até a folha de eventos e clique em `Add action` para adicionar uma ação:

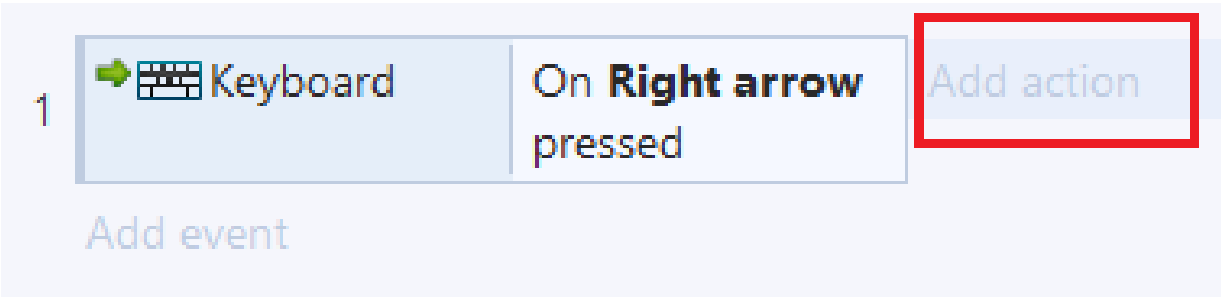


Figura 4.20: Adicionando uma ação

Na janela que abrir, selecione o `JogadorAnimacao` e escolha a ação `Set mirrored` (setar espelhamento). Na aba aberta, escolha a opção `Not mirrored` (não espelhado) e clique em `Done`:

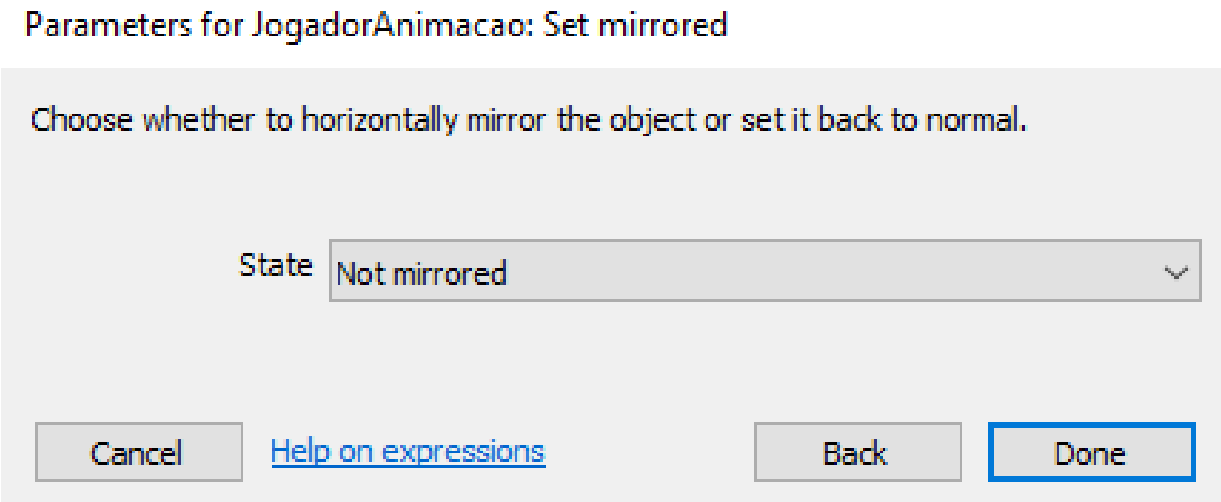


Figura 4.21: Adicionando uma ação ao jogador

Como o nosso boneco já está virado para a direita por padrão, colocamos a opção para que ele não reflita nada. Para a tecla da esquerda, temos de fazer diferente. Adicione a condição da tecla da esquerda assim como fizemos com a da direita, e insira a ação de `Mirrored`. Ela deverá ficar assim:

Figura 4.22: Condições e ações das teclas

Nesses dois eventos, cada vez que a tecla da seta direita for pressionada, o jogador virará para a direita (espelhar) e, quando clicada a tecla da esquerda, ele vai virar para a esquerda. Bem legal, né? Mas ainda falta uma animação, a de pulo. Então, vamos lá.

Clique em `Jogador` na janela de eventos e, em seguida, escolha a condição `Is Jumping` (está pulando). Para a ação, selecione o `JogadorAnimacao` e escolha a opção `Set Animation` (setar animação). Procure pela animação "pulando" que criamos anteriormente e clique em `Done`:

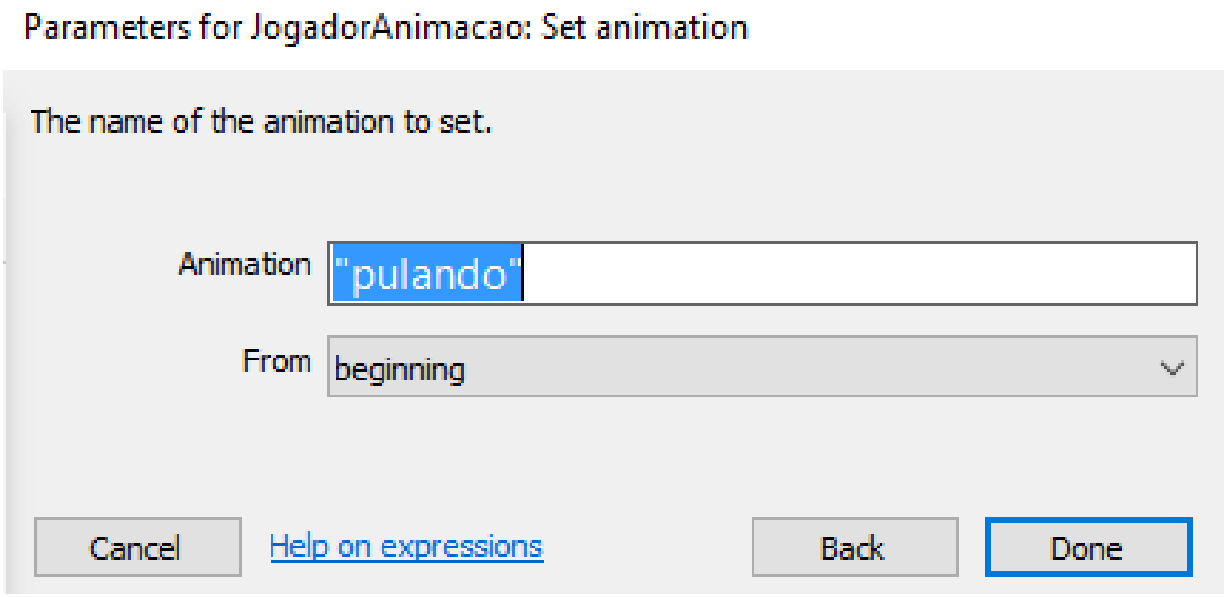


Figura 4.23: Animação "pulando"

Pronto! O único detalhe que falta é tratarmos a animação para quando o personagem estiver completamente parado, ou seja, o jogador não está pressionando nada no teclado. Na janela de eventos, selecione o `Jogador` e dê a condição `On landed` (parado).

Para conseguir fazer a validação de não estar apertando nada, vamos ter de usar os subeventos. Eles nada mais são do que os eventos que já estamos estudando, com diferença de que estes estão associados a um outro evento "pai". Vamos utilizá-los para podermos testar mais condições nos nossos eventos.

Para adicionar um subevento na nossa condição anterior, clique com o botão direito do mouse no canto esquerdo da condição. Selecione a opção `Add` e, em seguida, `Add sub-event` (ou clique em `S` do seu teclado). Para esse subevento, selecione outra vez o

objeto `Jogador` e escolha a condição `Is moving` (movendo) e, para a ação, selecione o objeto `JogadorAnimacao` e escolha o valor "andando" em `Set Animation`.

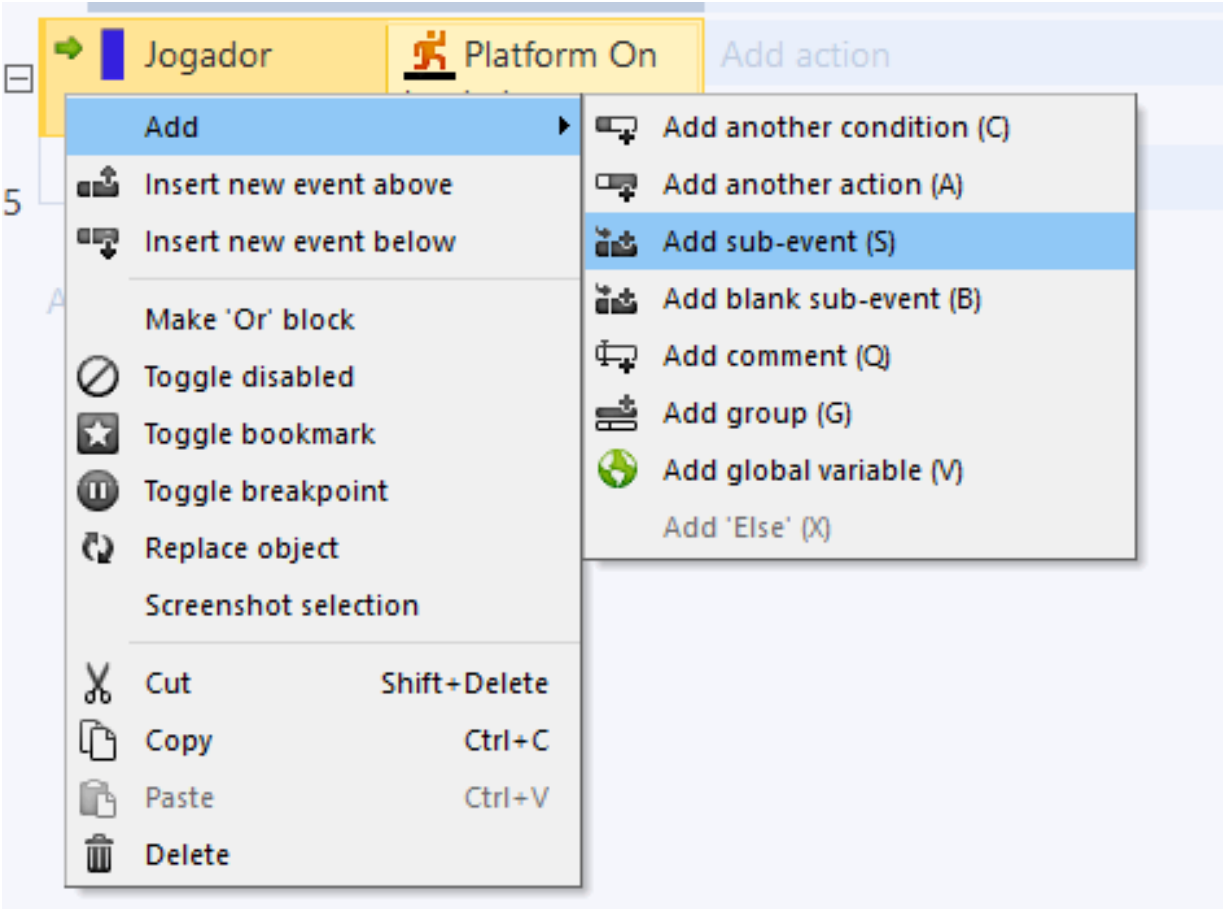


Figura 4.24: Adicionando um sub-event

Precisamos setar a animação do jogador para "parado" para quando ele não estiver se movendo. Para isso, adicionaremos uma condição chamada `Else` (se não) no nosso evento. Ela será sempre ativada quando a condição anterior não for respeitada.

Clique no canto esquerdo do subevento com o botão direito do mouse, selecione a opção `Add` e, depois, `Add Else` (ou aperte a tecla `X`). Na condição `Else`, selecione o objeto `JogadorAnimacao` e escolha o valor "parado" em `Set Animation`. Isso fará com que, quando o jogador estiver se movendo, ele tenha a animação; caso contrário, fique parado.

Sua folha de eventos ficará assim:

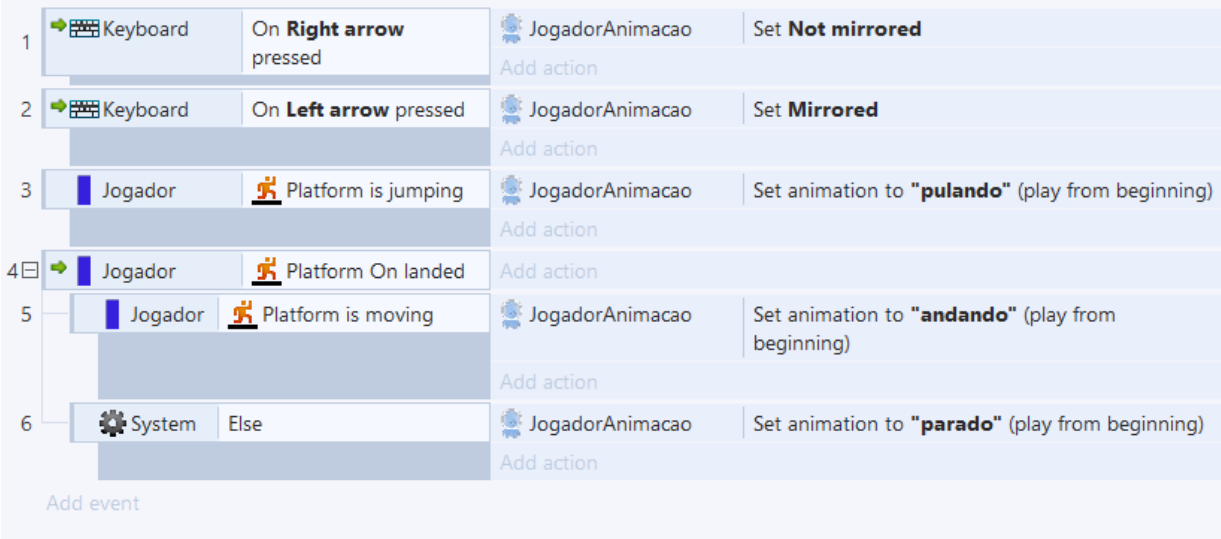

1 Keyboard On Right arrow pressed JogadorAnimacao Set Not mirrored
Add action
2 Keyboard On Left arrow pressed JogadorAnimacao Set Mirrored
Add action
3 Jogador Platform is jumping JogadorAnimacao Set animation to "pulando" (play from beginning)
Add action
4 Jogador Platform On landed Add action
5 Jogador Platform is moving JogadorAnimacao Set animation to "andando" (play from beginning)
Add action
6 System Else JogadorAnimacao Set animation to "parado" (play from beginning)
Add action
Add event

Figura 4.25: Folha de eventos

Agora vamos adicionar mais dois eventos para setar a animação do jogador parado (`stopped`) ou se movendo (`moved`), assim terminaremos a sua animação.

Para isso, adicione um novo evento para o objeto `Jogador` e escolha a condição `On Stopped` para ele. Para a ação, selecione o objeto `JogadorAnimacao` e escolha o valor "parado" em `Set Animation` . Para o último evento, selecione novamente o objeto `Jogador` e escolha a condição `On moved` . E para a ação, selecione o objeto `JogadorAnimacao` e escolha o valor "andando" em `Set Animation` .

Sua folha de eventos deve estar assim:

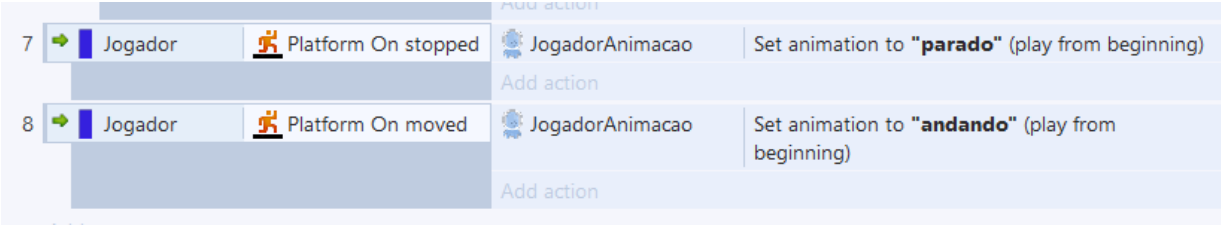

7 Jogador Platform On stopped JogadorAnimacao Set animation to "parado" (play from beginning)
Add action
8 Jogador Platform On moved JogadorAnimacao Set animation to "andando" (play from beginning)
Add action

Figura 4.26: Eventos da animação do jogador

Pronto, temos todos os eventos de movimento e animação do nosso personagem programados. Você se lembra de que faríamos a base do jogador e suas animações de forma separadas? E que, pela folha de eventos, faríamos que elas sempre ocupassem a mesma posição, dando a impressão que é um objeto só?

Chegou a hora de fazer isso para finalmente finalizarmos a animação do nosso personagem. Como faremos isso? Usaremos as funções `Set position` (setar posição) e `Pin` (fixar). Com elas, podemos especificar a posição exata do objeto dentro do plano cartesiano, ou seja, utilizando as coordenadas X e Y.

Para implementar isso, adicione um novo evento na nossa folha. Selecione o `System object` e procure pela condição `On start of layout` (no início do layout). Isso fará com que todas as ações que especificarmos sejam executadas assim que o layout for carregado.

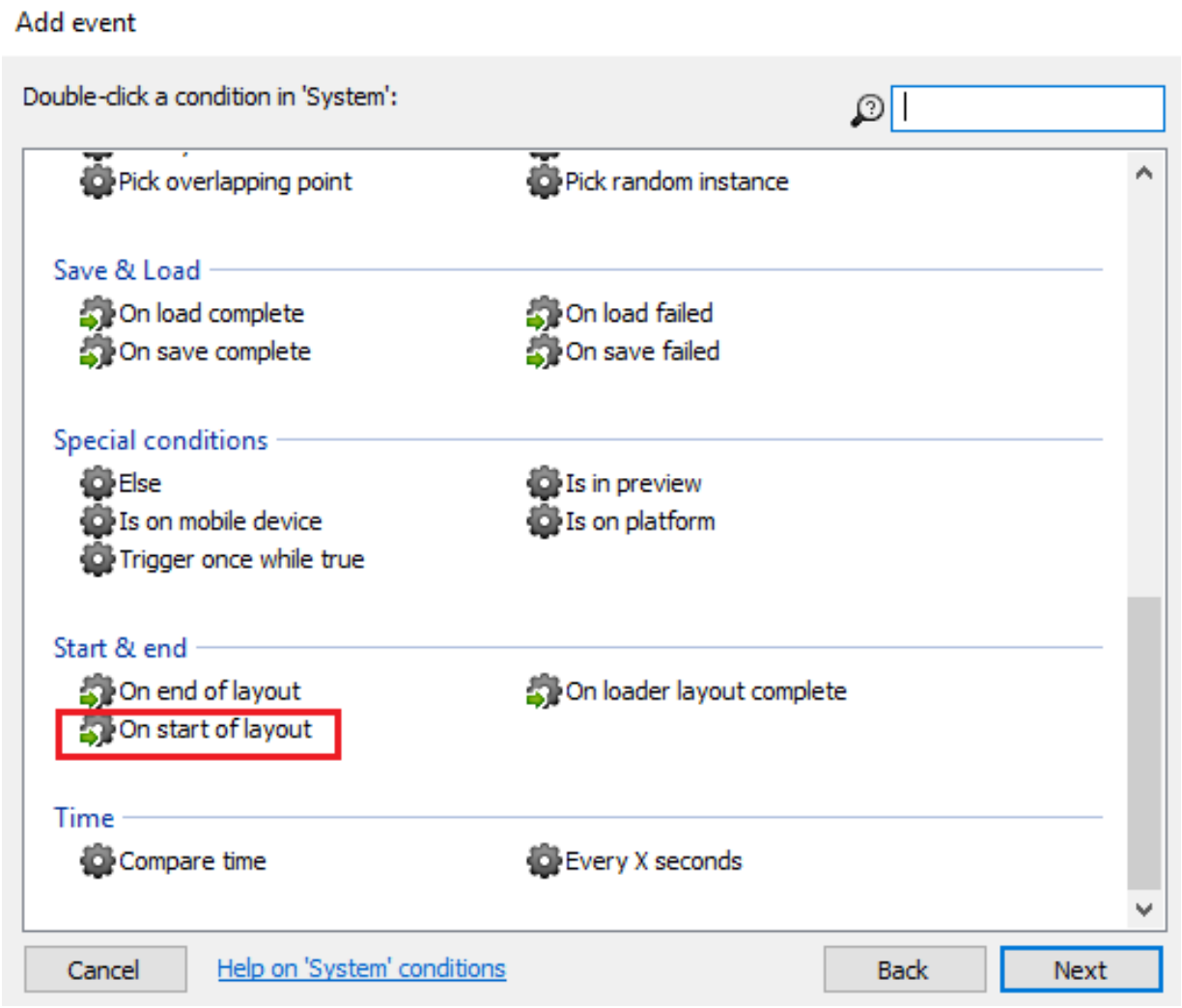


Figura 4.27: Quando inicializar o layout

Para a ação dessa condição, vamos selecionar o objeto `JogadorAnimacao` e escolher a opção `Set position` (Setar posição). Assim será possível escolhermos a posição X e Y onde queremos posicionar o nosso personagem.

Como desejamos que a animação comece sempre na mesma posição da base do jogador, vamos utilizar como referência as coordenadas dele. Adicione os seguintes valores para X e Y, que serão a posição do nosso `Jogador` :

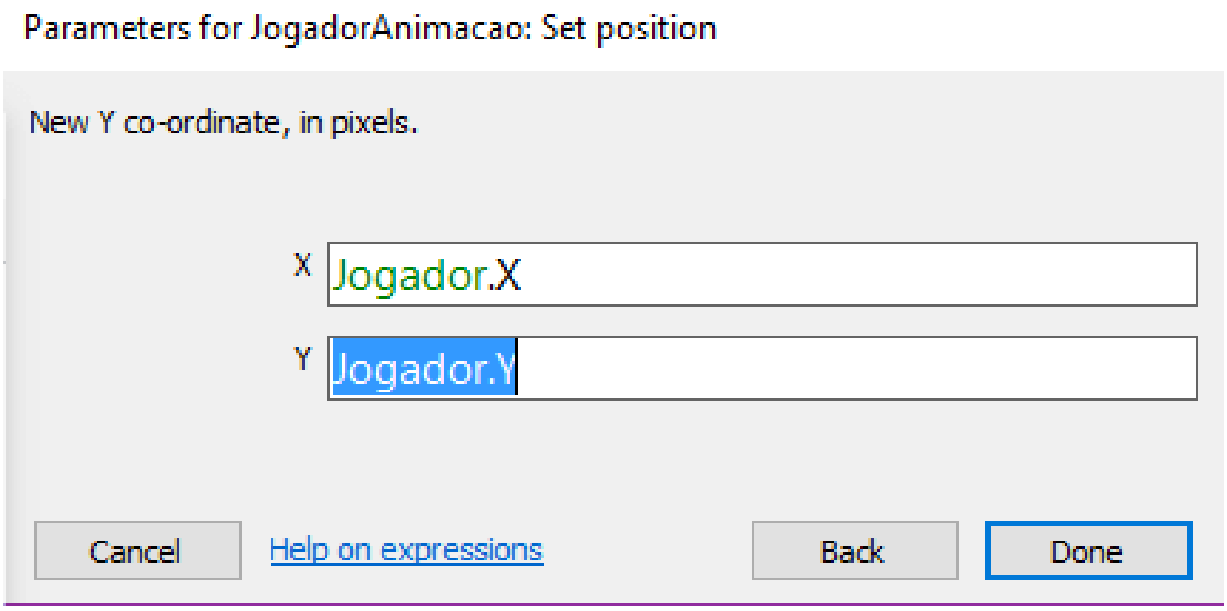


Figura 4.28: Adicionando a posição do jogador

Vamos adicionar mais uma ação nessa condição de `On start of layout` para "colar" o nosso `JogadorAnimacao` no `Jogador`, pelo comportamento `Pin` adicionado anteriormente. Isso fará com que a animação fique colada à base do jogador, de modo que sempre vai acompanhá-lo. Sem isso, ao mover a sua base, o objeto de animação ficaria no mesmo lugar.

Então, escolha novamente `JogadorAnimacao` na janela `Add event`, e selecione a ação `Pin to object`, como a seguir:

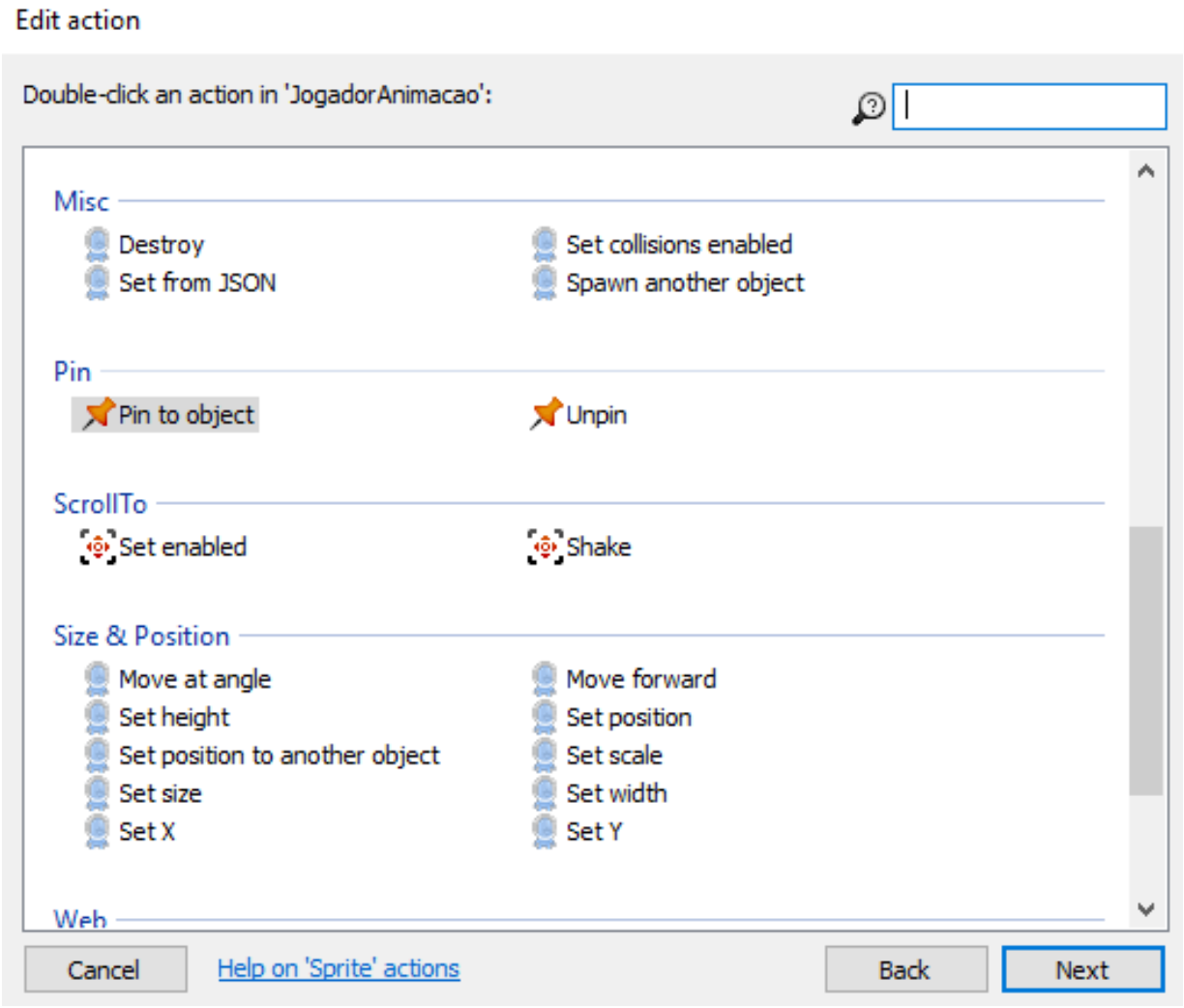


Figura 4.29: Adicionando a ação de Pin

Na aba que abrir, escolha o objeto `Jogador` na opção `Pin to` e clique em `Done`:

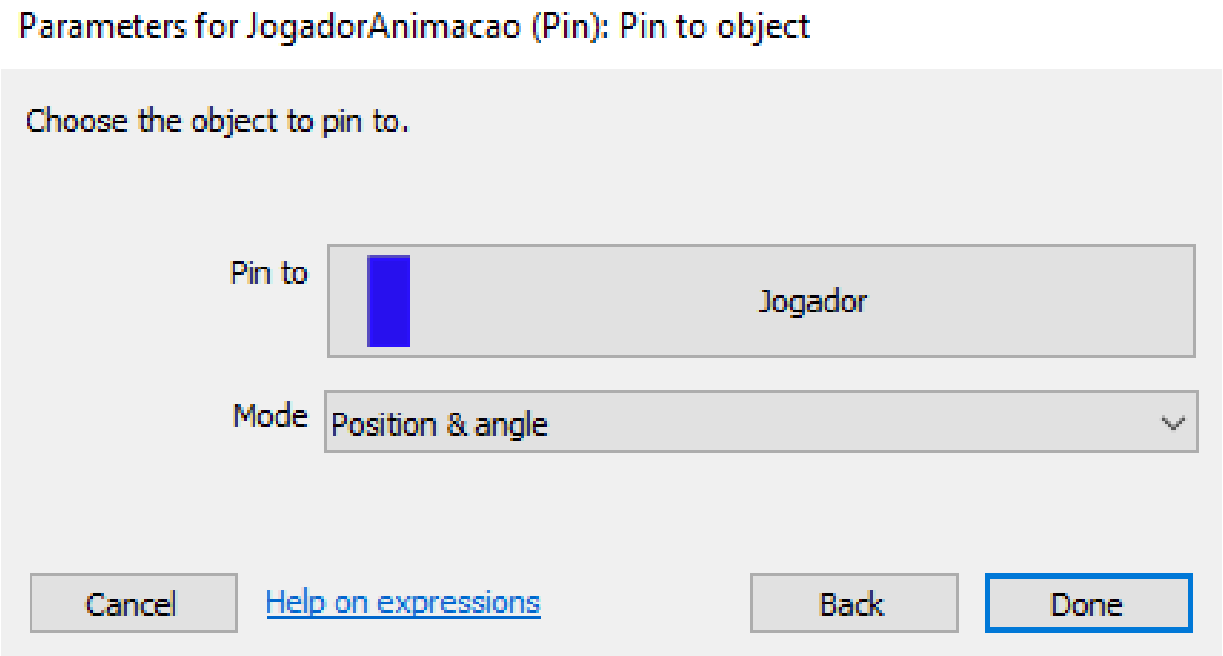


Figura 4.30: Pin to object

Esse último evento criado ficará assim:

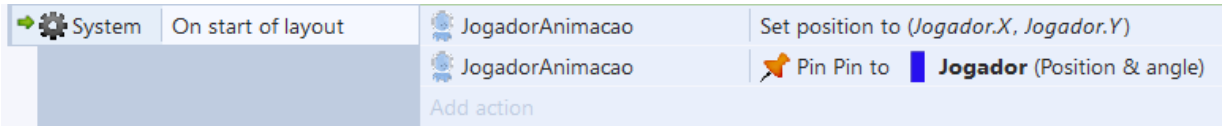


Figura 4.31: Evento On start of layout

Ufa, finalmente estamos prontos! Agora que terminamos de posicionar o nosso `JogadorAnimacao` no layout, já podemos testar e ver como nosso jogo está ficando legal! Para testar, vá até a barra superior na aba `Home` e escolha a opção de `Run layout`, como já explicado no capítulo *Conhecendo a plataforma*. Clique nas setas do teclado para ver o jogador se movendo.

Capítulo 5
Sistema de vida e adição de inimigos

No capítulo anterior, fizemos um ótimo trabalho ao adicionar movimentação e animações ao nosso personagem. Passamos por conceitos importantíssimos do Construct 2, desde o editor de imagem e seus menus para animação (envolvendo o uso de quadros por segundos) até os primeiros eventos dentro das folhas de eventos.

Utilizamos os eventos de `Keyboard` (teclado) para associar as animações aos comandos dados pelo jogador. Observamos que os eventos possuem basicamente duas partes: as condições e as ações. Além disso, também usamos subeventos para conseguir descrever condições mais específicas para a animação de cada um dos estados do nosso personagem: parado, andando (direita e esquerda) e pulando.

Neste capítulo, vamos ver outro conceito importantíssimo para o desenvolvimento de jogos dentro do Construct 2, as variáveis. Para mostrar como usá-las, levaremos o nosso jogo para o próximo patamar, adicionando um sistema de vida.

Esse sistema fará com que o nosso jogo termine se o jogador ferir o nosso personagem mais de três vezes. A seguir, adicionaremos os inimigos e associaremos esse sistema ao comportamento deles.

Preparados? Então, vamos nessa.

5.1 Adicionando vidas ao personagem

Como dissemos no início do capítulo, nossa missão aqui será adicionar um sistema de vidas para o nosso personagem, dando a

possibilidade de o jogador perder o jogo. Faremos com o que ele inicie com três vidas, que serão representadas por corações na tela (não se preocupe, pois faremos todos os passos juntos).

A ideia é que cada vez que ele colidir com um inimigo, ou sair da área visível do layout, ele perca uma vida. Uma vez que ele perca todas as três, a tela de final de jogo aparecerá e este será finalizado – "finalizado" aqui significa que o jogador perderá todo o progresso e só terá a possibilidade de iniciar o jogo do início. Para fazer tudo isso, aprenderemos um conceito muito importante, o de variáveis.

Quando falamos de computadores, variáveis são espaços na memória RAM (*Random Access Memory*) da máquina física. Essa máquina pode ser um computador desktop, um notebook, um tablet ou mesmo o seu smartphone. Nesses espaços, podemos armazenar qualquer tipo de informação, desde números, palavras ou até textos inteiros.

Todo programa (ou aplicativo) que roda no seu computador utiliza dezenas, centenas ou até milhares de variáveis na memória. É por isso que, quanto maior ela for, mais coisas você consegue fazer simultaneamente.

Este espaço de memória fica fora do seu disco rígido, pois ela é uma memória de rápido acesso. Quando um programa é carregado, o sistema joga-o para dentro desta memória, e o processo usa as variáveis lá dentro para controlar o seu fluxo. E com o Construct 2, não é diferente.

Ao abrir um jogo, ele também é jogado para esta memória, e podemos utilizar variáveis para controlar o seu fluxo. Dentro do Construct, podemos declarar dois tipos de variáveis: locais e globais. Cada uma tem contexto e comportamento diferentes. Vamos ver com mais detalhes.

Variáveis locais

O primeiro tipo de variável que veremos são as locais – também chamadas de variáveis de instância. Elas possuem este nome porque estão sempre associadas ao objeto que as declararam. Mas o que isso significa?

Quer dizer que podemos ter uma mesma variável em dois objetos diferentes, com dois valores diferentes. Usamo-las para guardar informações que sejam relevantes para o objeto que as contêm, como o número de vidas deste objeto. Será que você entendeu aonde queremos chegar? Vamos ver como funcionam as variáveis locais na prática.

Começaremos adicionando uma variável local ao nosso objeto `JogadorAnimacao`. Para isso, selecione-o e vá até as suas propriedades (apresentadas na `Barra de Propriedades`). Encontre a seção `Instance variables` (variáveis de instância), como mostra a figura a seguir. Nesta opção, declaramos as variáveis que pertencerão a este objeto.

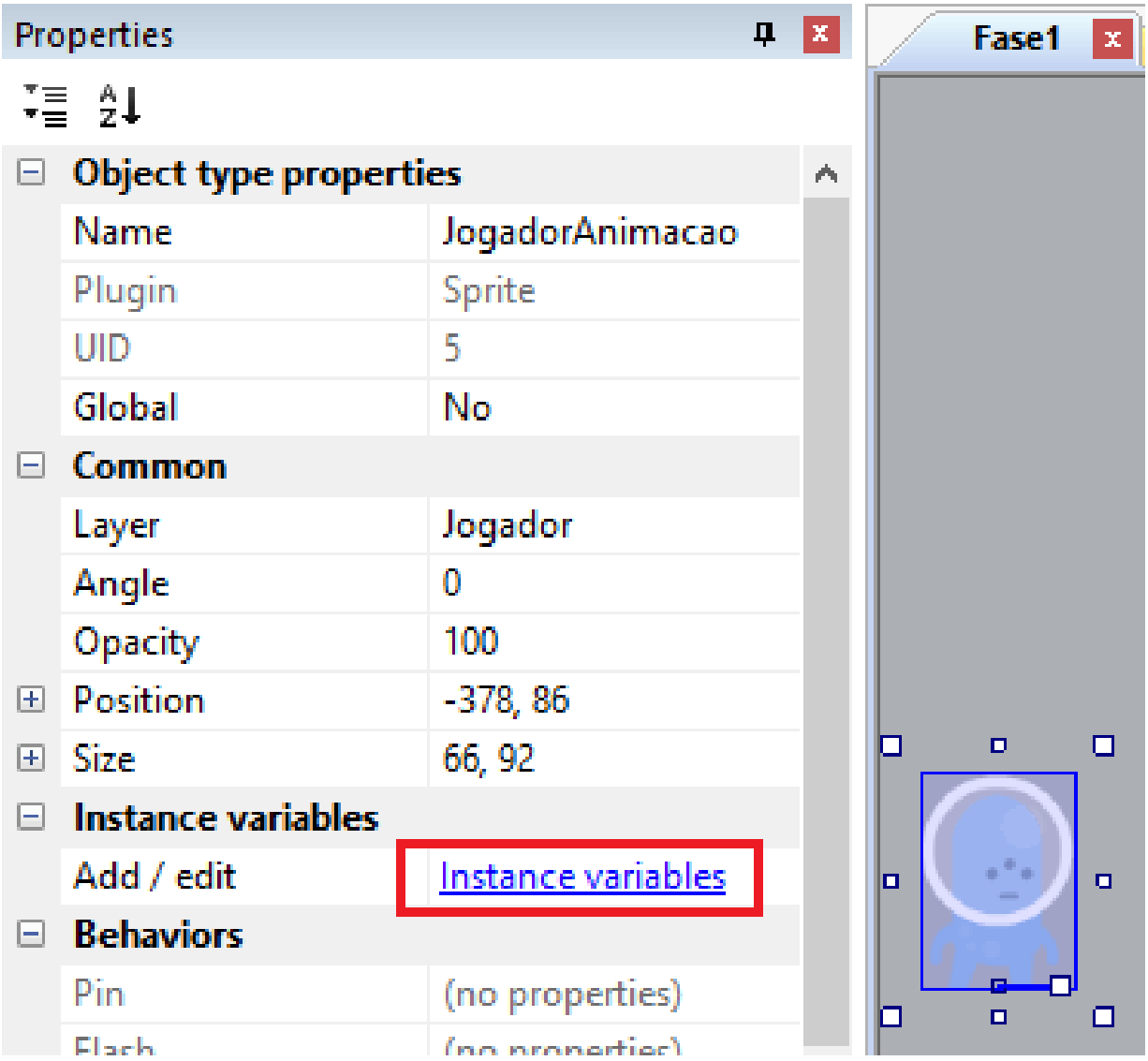


Figura 5.1: Instance variables

Na coluna `Add/edit` (adicionar/editar), clique no link onde está escrito `Instance variables`. Uma pequena janela vai se abrir oferendo algumas opções. Procure pelo símbolo de soma (`+`) e adicione uma variável nova. Na próxima janela, você encontrará quatro campos:

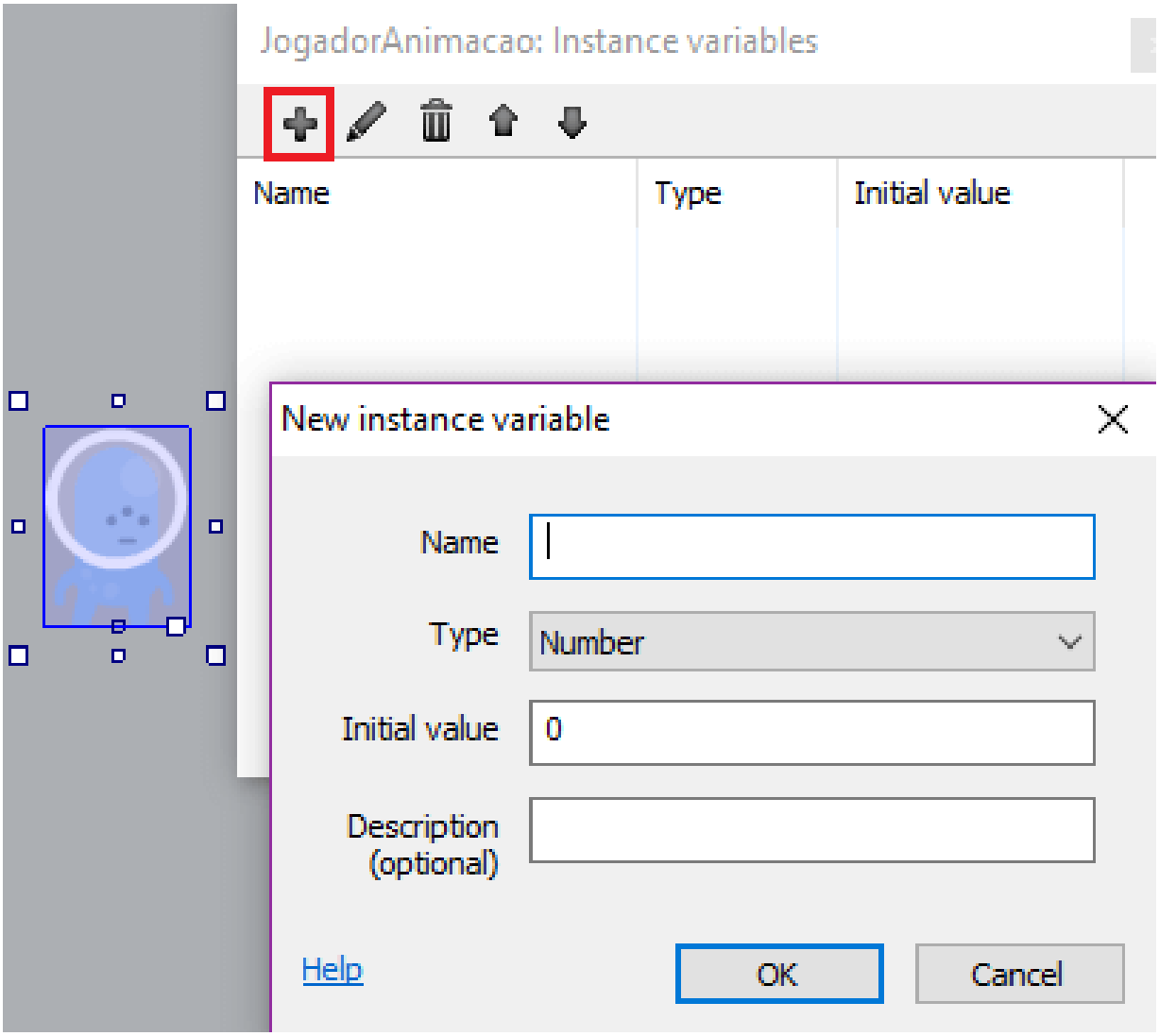


Figura 5.2: Adicionando uma variável local

- *Name* (nome);
- *Type* (tipo);
- *Initial value* (valor inicial);
- *Description* (descrição).

Como os próprios nomes dos campos já são elucidativos, não fica complicado entendermos para que servem. No campo `Name`, inserimos o nome que queremos para a variável. Para o nosso caso, coloque `vida`. No segundo campo, o tipo (`Type`), temos três opções possíveis:

- **Boolean (booleano):** este tipo só assume dois valores, verdadeiro (*true*) ou falso (*false*).
- **Number (número):** tipo específico para variáveis que representam números racionais.
- **Text (texto):** tipo que guarda palavras, textos ou mesmo um caractere.

Como estamos tratando o número de vidas do nosso personagem, escolheremos o tipo `Number`, que é o que faz mais sentido nessa situação. Mas para que você saia deste capítulo com total domínio dos tipos de variáveis que podemos criar, vamos explicar rapidamente como funcionam os outros.

O booleano representa apenas dois valores: verdadeiro (*true*) ou falso (*false*). Esse tipo de variável tem origem lá na física, em que tudo é representado desta maneira (verdadeiro é ligado, e falso é desligado). Dentro do Construct 2, podemos utilizá-la de muitas formas. Um exemplo simples seria para saber se uma alavanca foi acionada pelo jogador. Poderíamos criar uma variável chamada `acessado` com o valor `false` por padrão, e somente trocá-la quando ela fosse acionada. Desta forma, ficaria fácil saber e usar essa informação para outras coisas no jogo.

Já o tipo texto não tem segredo. Utilizamos este tipo sempre que precisamos trabalhar com textos dentro dos nossos jogos. Por exemplo, se quisermos exibir o nome do jogador dentro do jogo, podemos armazenar essa informação em uma variável do tipo texto e usá-la em qualquer lugar do jogo. Bem bacana, né?

Exploraremos um pouco mais sobre as variáveis e seus tipos no decorrer do livro, mas agora é importante que você já sabe para que elas servem. Voltando à nossa variável `vida`, escolha o tipo `Number`.

Em seguida, temos o campo `Initial value`, em que especificamos o valor inicial para a variável. Como desejamos que o jogador inicie com três vidas, qual número colocaremos neste campo? Isso mesmo, o número três.

Por fim, o último campo é totalmente opcional. Ele serve para colocar alguma descrição na variável, para que futuramente você (ou seus colegas de trabalho) não esqueça o que ela faz e para que serve.

Particularmente, não usamos muitos este campo, porque sempre preferimos criar nomes que sejam elucidativos, ou seja, que consigam descrever a sua razão, sem qualquer tipo de comentário adicional. Esta é uma prática que herdamos do mundo da computação e recomendamos fortemente, mas você está livre para trabalhar da maneira que achar melhor, ok?

Ao final deste processo, sua variável deverá ficar igual à figura:

Figura 5.3: Variável local vida

Muito bem, agora que criamos esta variável, a usaremos para saber quantas vidas o nosso personagem ainda tem. No início do jogo, ela será iniciada com três. Se levar dano de um inimigo, subtrairemos uma. Se sair da área visível, também subtrairemos uma. Ao final, se a contagem da variável for igual a zero, vamos exibir a tela de *game over* e o jogador saberá que perdeu. Bem legal, né?

Mas antes de partir para o desenvolvimento de toda essa lógica, queremos introduzir as variáveis globais. Elas são parecidas com as

que acabamos de ver, mas não estão "fixas" a um objeto, e sim ao contexto da folha de eventos em que foi criada. Ficou curioso?

Variáveis globais

Uma variável global – como o próprio nome já diz – engloba todo o escopo do jogo. O que isso significa na prática? Significa que todos os objetos e eventos podem interagir com ela, como se fossem suas próprias donas. Elas são apropriadas para guardar informações que sejam de contexto geral do jogo, como estatísticas. No nosso caso, vamos usá-la em outro caso recorrente: para guardar a pontuação do jogador.

Neste ponto, você deve estar se perguntando como saber quando usar um tipo de variável e quando usar o outro. Queríamos ter uma resposta concreta para isso, mas infelizmente não há. Tudo depende de como você quer organizar o seu jogo.

Com base em nossa experiência, utilizamos as globais somente para guardar informações que realmente sejam relevantes para o jogo como um todo, como pontuações e rankings. Por ora, siga os nossos passos e, quando se sentir confortável com a plataforma, tente outras maneiras e veja qual melhor se adapta a você.

Seguindo com o desenvolvimento do nosso jogo, vamos armazenar a pontuação do jogador em uma variável global. A pontuação do nosso jogo será feita através dos itens coletáveis que vamos inserir mais à frente. Por enquanto, vamos apenas deixar o terreno preparado.

Criaremos nossa variável na folha de eventos. Para isso, clique com o botão direito do mouse nela e selecione a opção `Add global variable` (Adicionar uma variável global). Uma janela se abrirá com o nome de `New global variable` (Nova variável global), como mostra a figura a seguir.

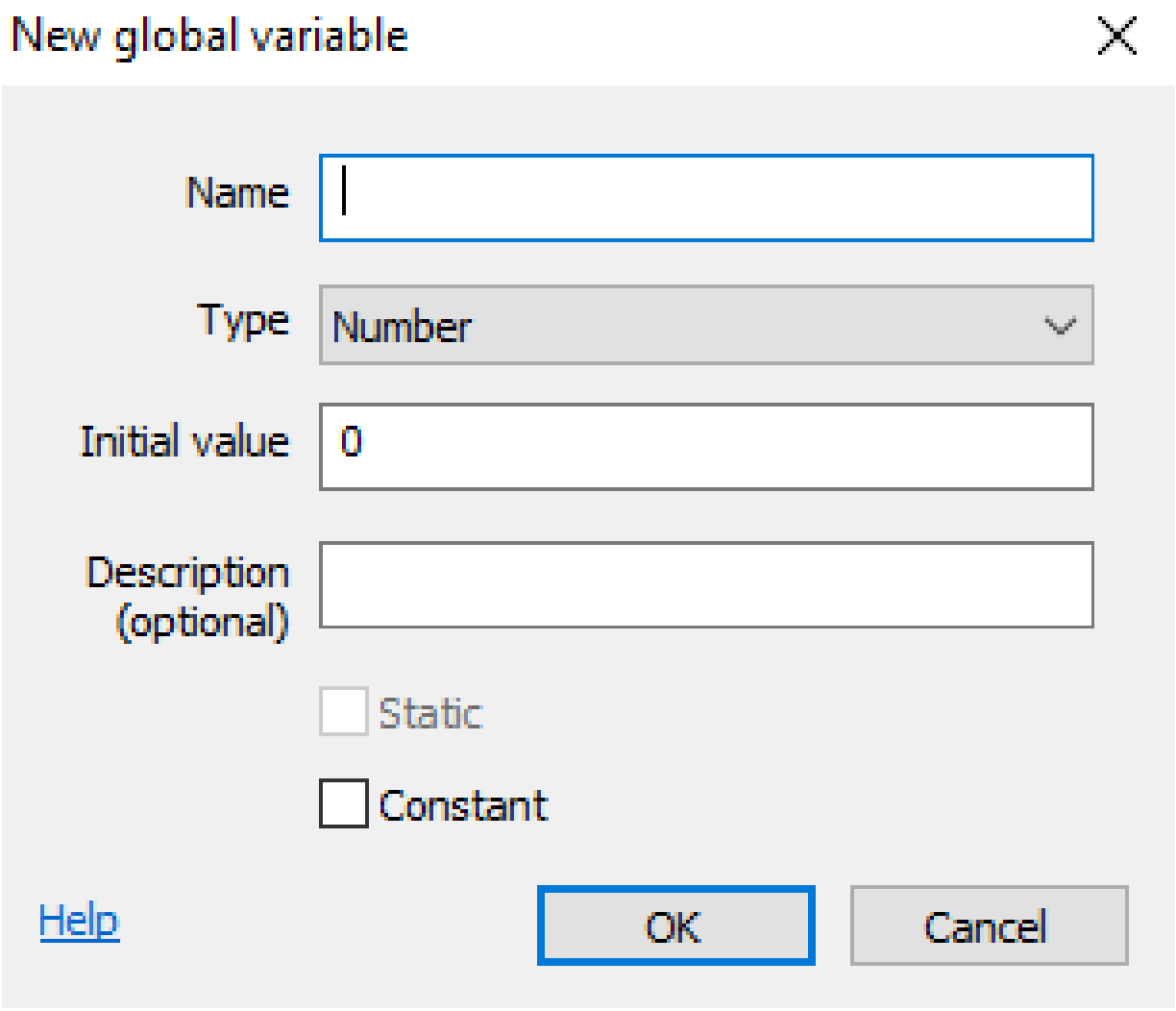


Figura 5.4: Adicionando uma variável global

Você vai reparar que ela é muito semelhante à janela que vimos na seção anterior, mas que há algumas diferenças. A primeira delas é que agora só podemos criar variáveis de dois tipos: *Number* e *Text*. A segunda é que, depois do texto de descrição, há duas caixas com as opções `Static` (estático) e `Constant` (constante).

As variáveis que são marcadas como estáticas não reiniciam os seus valores quando entram dentro do seu escopo. Seus valores são persistidos, tornando-as globais. Não se preocupe muito em entender isso agora, discutiremos isso mais à frente quando falarmos sobre grupos. Por enquanto, saiba que toda variável global

é marcada automaticamente como estática. Há alguns segredos sobre variáveis que vamos esconder de você, mas só por enquanto.

A segunda caixa torna as variáveis para somente leitura (*read only*), ou seja, não conseguimos nunca alterar seus valores, somente ler e compará-los com outros. Você pode se perguntar: "Mas em que contexto vou querer uma variável que não muda de valor?".

Sabemos que isso parece meio estranho à primeira vista, mas elas são bastante úteis, por exemplo, para gravarmos valores que são referências para outras tarefas. Poderíamos armazenar o valor da força da gravidade nela e fazer com que tudo fosse calculado de acordo com isso. Ou poderíamos armazenar o valor de uma constante para usar dentro de uma fórmula matemática, como o PI.

Há vários contextos em que elas seriam úteis. Mas fique tranquilo, não usaremos variáveis constantes. Entretanto, tenha em mente a sua utilidade.

Com a janela de criação de variáveis, preencha o campo `Name` com o valor `pontos`, o campo `type` com a opção `Number`, e coloque o número zero como `Initial value`. Fique à vontade para adicionar uma descrição e deixe as caixas a seguir desmarcadas. Ao final, ela deve ficar desta maneira:

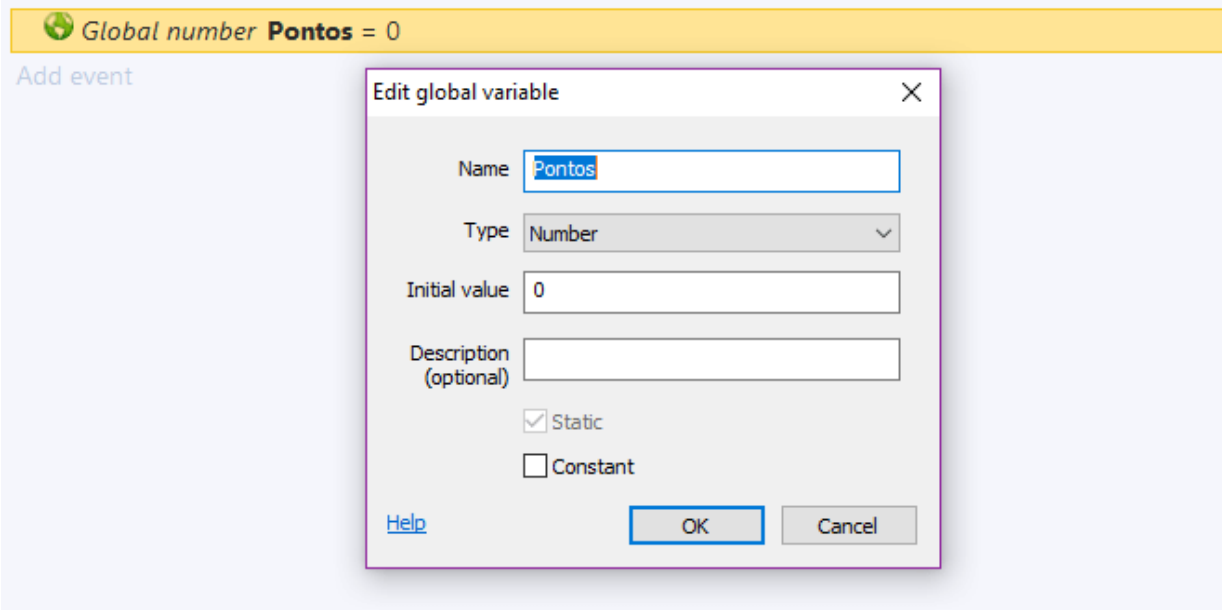


Figura 5.5: Variável global

Muito bem! Preparamos o sistema de vidas, e iniciamos o sistema de pontuação do nosso jogo utilizando as variáveis locais e globais. Nosso próximo passo será inserir os inimigos no jogo.

5.2 Inimigos

Agora que já temos o nosso personagem com movimentos, animações e vidas, vamos começar a dificultar a vida dos nossos jogadores colocando inimigos no jogo. Para este projeto, planejamos três tipos de inimigos, mas, após concluí-los, fique à vontade para deixar sua criatividade fluir e adicionar mais tipos. Por ora, siga os nossos passos, ok?

Então, ao adicionar inimigos, precisamos criar uma inteligência de movimentação e configurar cada uma de suas ações em relação ao personagem. Queremos que, quando o personagem encoste neles, perca uma vida e sinalize que recebeu dano.

Entretanto, nosso personagem não será totalmente indefeso. Também configuraremos que, se ele acertar os inimigos por cima, eles morrem. Vamos nessa?

Inserindo os inimigos

Para começar o desenvolvimento dos inimigos, adicione uma nova `Sprite` na camada `Jogador` com o nome de `inimigo1`. Agora, para ajustarmos sua animação (assim como fizemos com o personagem principal), adicione dois quadros para a animação padrão. Ao criar os quadros, importe as imagens de nome `bolinha1` e `bolinha2` que estão no diretório chamado `Inimigos`, dentro da pasta de recursos do livro.

Feito isso, nosso inimigo terá uma animação, mas ela está estranha, pois a sua velocidade padrão é muito alta para apenas dois quadros. Para ajustá-la, vá até a barra de propriedade e altere a propriedade `Speed` (velocidade). Mude o seu valor para três.

Por fim, vamos fazer com que a animação se repita para sempre enquanto o inimigo estiver vivo. Para isso, procure pela propriedade `Loop` (repetição) e coloque o valor `Yes`.

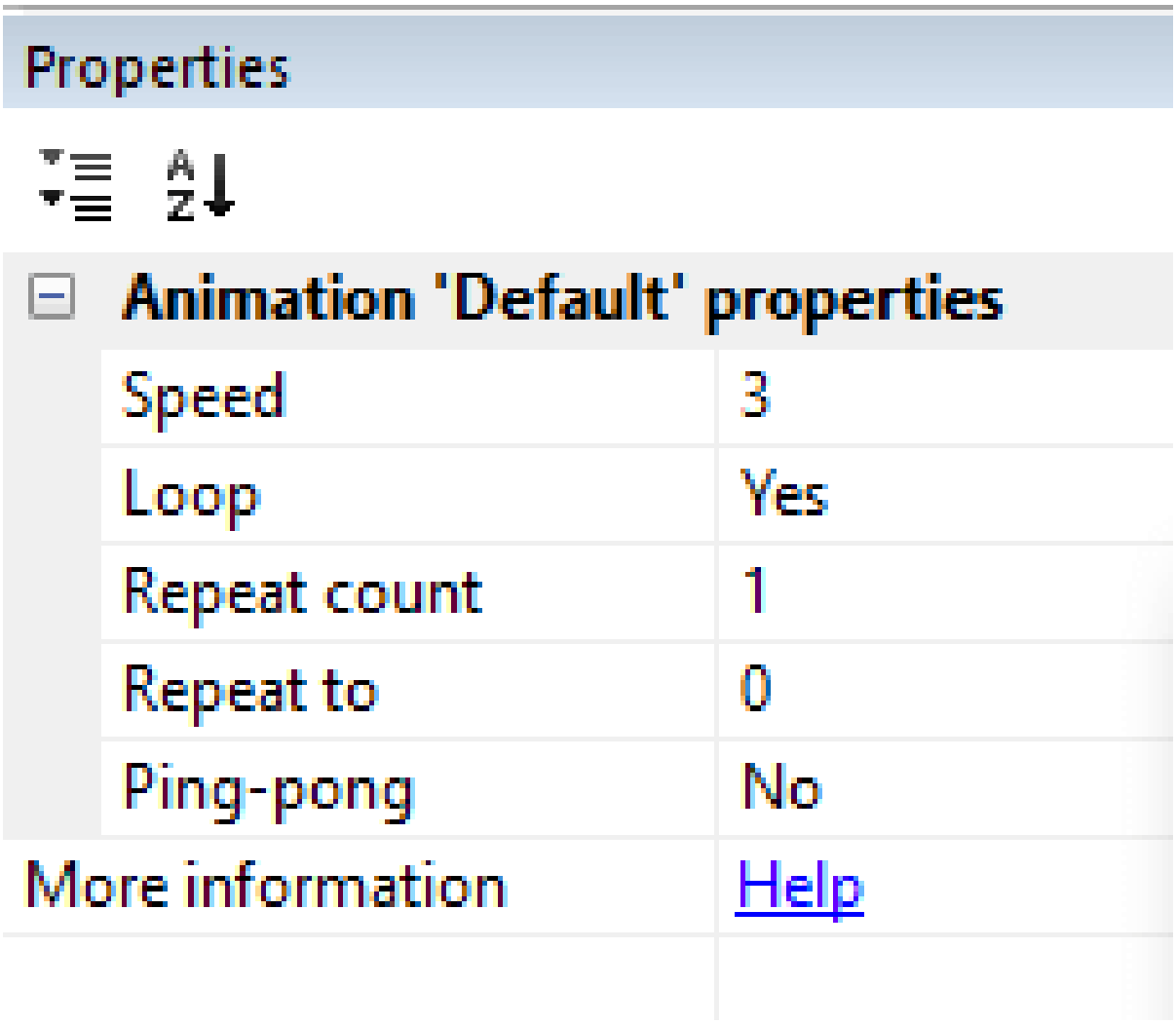


Figura 5.6: Velocidade da animação

Agora sim! Para verificar como ela está, clique com o botão direito em cima do seu nome (no caso, a `Default`), e selecione a opção `Preview`, como mostra a figura a seguir:

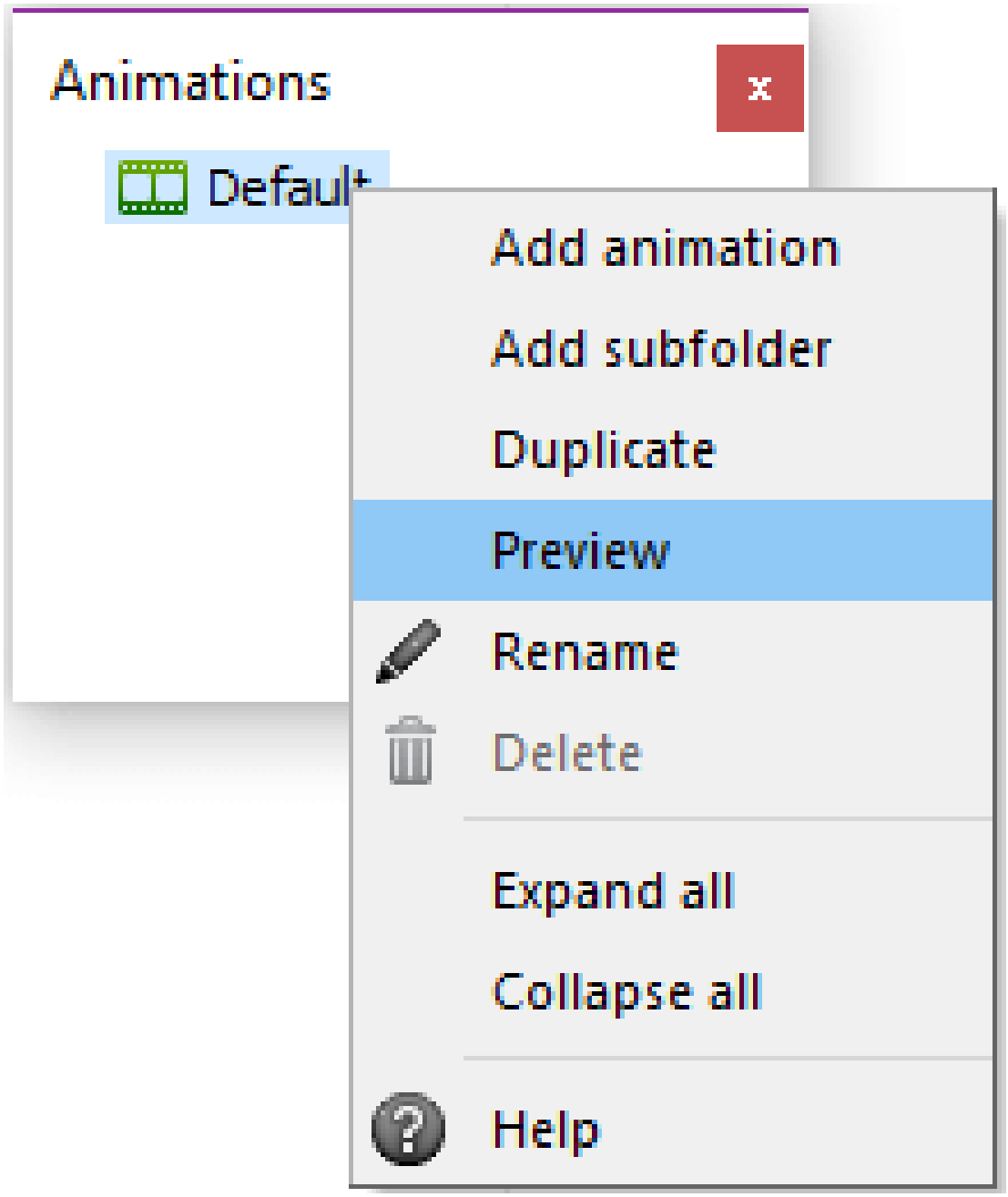


Figura 5.7: Selecionando Preview

Na pequena janela que abrir, veremos como está a movimentação do nosso primeiro inimigo do jogo.

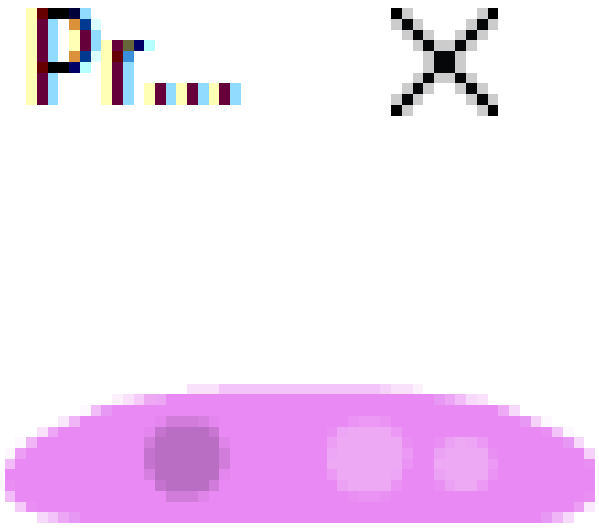


Figura 5.8: Preview da animação

Bem legal, né? Só não se esqueça de colocar o ponto de origem no centro e no meio (*bottom*) dos dois frames do `inimigo1`, como já fizemos com a maior parte dos itens que montamos até agora.

Agora repetiremos basicamente os mesmos passos para adicionar os dois outros tipos de inimigos restantes. Adicione uma outra `Sprite` para nosso outro inimigo com o nome de `inimigo2`. Novamente, insira dois quadros e coloque neles as imagens `peixe1` e `peixe2`, que também estão na pasta `Inimigos`. Configure as propriedades `Speed` e `Loop`. Por fim, adicione o ponto de origem no centro e no meio dos dois frames.

Para finalizar, faça o mesmo processo para o terceiro inimigo. Chame-o de `inimigo3` e use as imagens `abelha1` e `abelha2` nos quadros. As configurações das propriedades e o ponto de origem também devem ser os mesmos que os demais.

Figura 5.9: Os três tipos de inimigos

Muito bem! Colocamos os nossos três inimigos dentro do jogo. Mas está faltando alguma coisa, não é mesmo? Nossos inimigos ainda não têm vida nenhuma, são objetos estáticos. Na próxima seção, iniciaremos a programação de sua inteligência artificial.

Programando a inteligência dos inimigos

Já temos todos os inimigos inseridos no layout, agora podemos começar a programar sua inteligência. Você deve ter reparado, ao adicionar as imagens, que eles estão em posições diferentes. Enquanto os dois primeiros estão na horizontal, o terceiro está na vertical. Isso é proposital.

Faremos com que o segundo e o terceiro inimigos (ou seja, a abelha e o peixe) se movam na vertical, enquanto a pequena bolinha caminhe na horizontal. Ao final do processo, nossa abelha voará, o peixe pulará da água e bolinha vai rastejar pelo chão. Para alcançar isso, faremos o uso de comportamentos e variáveis.

Inicie selecionando o `inimigo1`, vá até a barra de propriedade e insira uma variável local – como já aprendemos anteriormente – do tipo `Number`. Coloque o nome `movendo` e o valor zero como padrão. Faça o mesmo para o `inimigo3`. Isso será útil para programar os eventos na próxima seção, já voltaremos a eles.

Agora selecione novamente o `inimigo1` e insira o comportamento de plataforma. Isso fará com que ele se mova pelo chão. Para o

`inimigo3`, insira o comportamento de `Sine`. Este move o objeto de acordo com o resultado de funções matemáticas, como já explicamos anteriormente. Adicione os seguintes parâmetros a seguir ao comportamento, para que ele se mova de forma vertical:

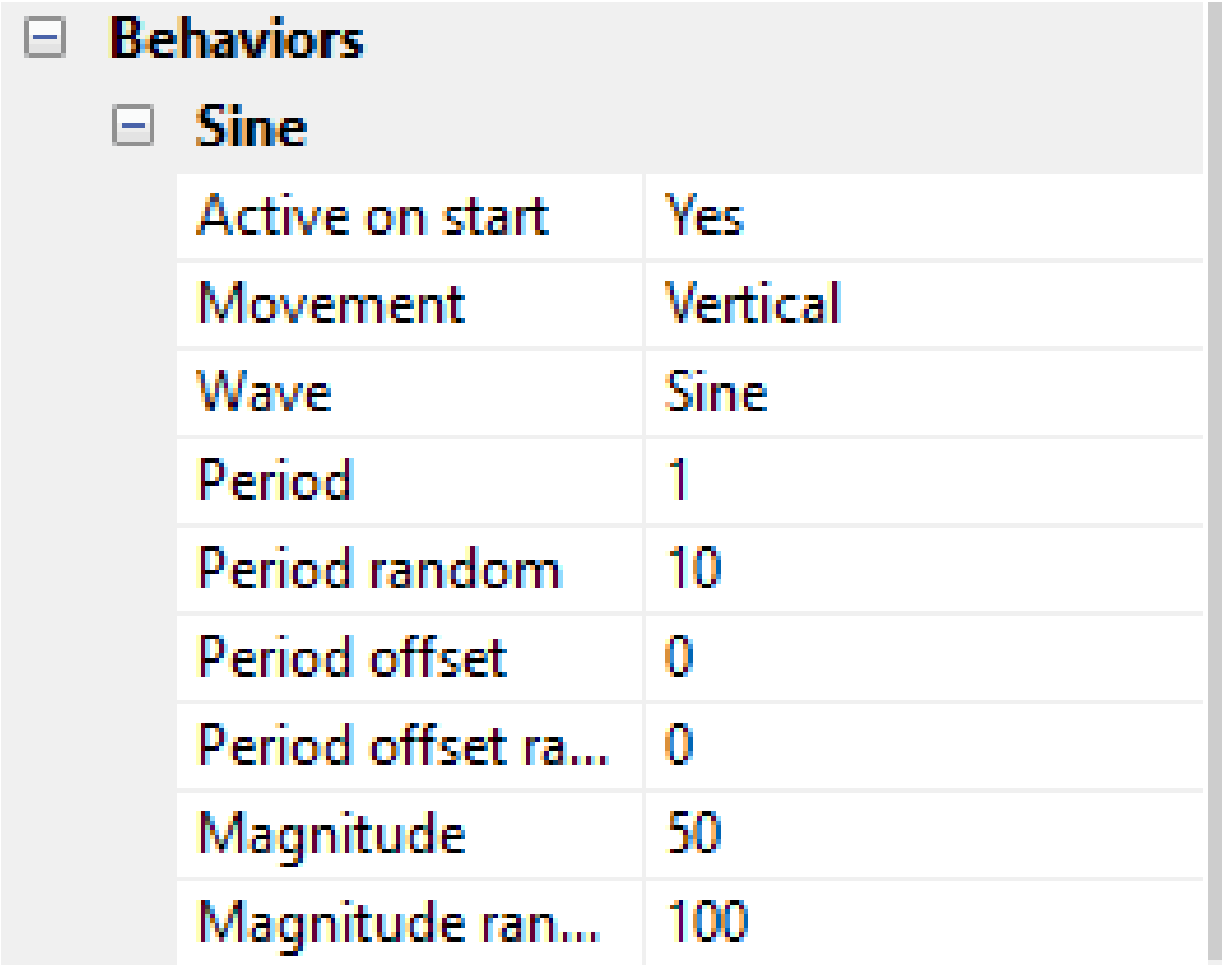


Figura 5.10: Behavior Sine no inimigo3

Maravilha, já deixamos a primeira parte da inteligência dos nossos inimigos pronta. Agora, vamos desenvolvê-los separadamente.

Programando o primeiro inimigo

Vamos adicionar os eventos referentes à movimentação do `inimigo1`. Para este, queremos que ele se mova no chão na direção

do jogador (à esquerda) assim que ele se tornar visível na tela. Ou seja, antes disso, ele deve esperar quietinho.

Para ter este comportamento, o primeiro passo é saber se ele está dentro da área visível do jogo. Faremos isso entrando na nossa folha de eventos e adicionando um novo. Selecione o `inimigo` e escolha a opção `Compare X` (comparar x). Com essa condição, conseguimos validar se a posição X do personagem está dentro da área visível.

Escolha esta opção e insira o valor `-100` (este valor não é mágico, estamos usando-o por causa do tamanho da janela que escolhemos para o jogo). Deverá ficar assim:

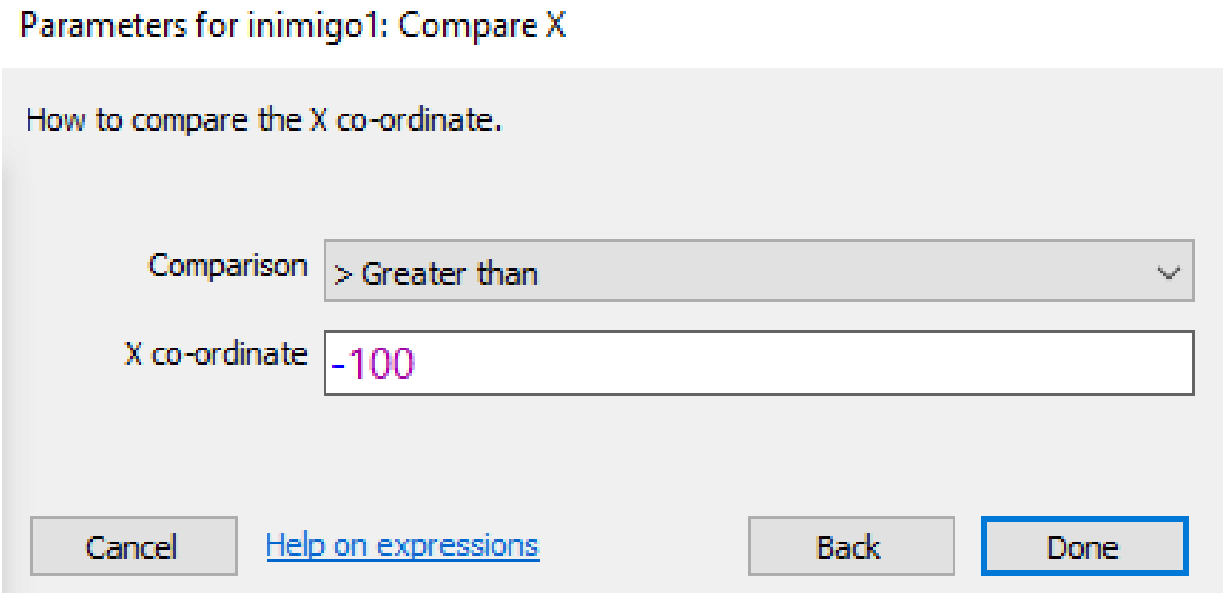


Figura 5.11: Comparando o valor de X

Colocaremos uma nova condição nesse mesmo evento, que verificará se o `inimigo1` está se movendo pela variável local `movendo` já criada. Vamos verificar da seguinte maneira: se a variável `movendo` for igual a `um` , o jogador estará se movendo; se for igual a `zero` , estará parado.

Para isso, clique no canto ao lado esquerdo e selecione a opção `Add` (adicionar) e, em seguida, `Add another condition` (adicionar nova

condição). Selecione novamente o `inimigo1` e escolha a condição `Compare instance variable` (comparar variáveis de instância). Isso fará com que possamos comparar o valor de variáveis locais.

No caso, queremos ver se a variável local `movendo` é igual a `um`. Para isso, coloque os parâmetros a seguir:

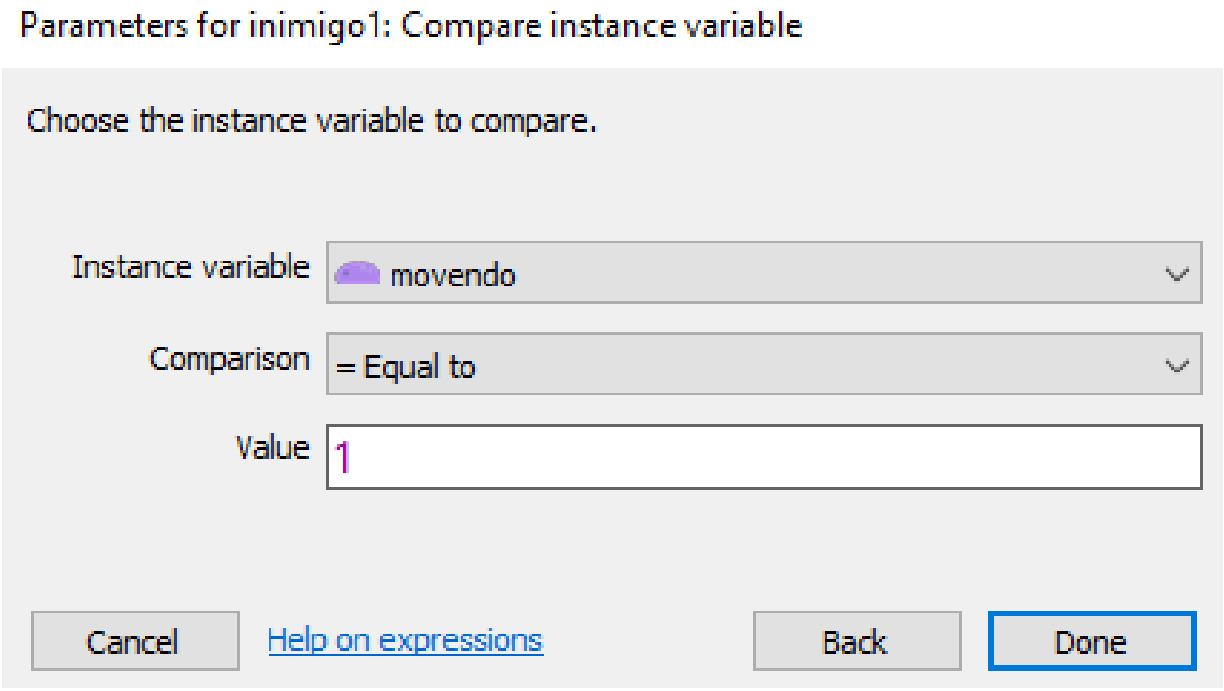


Figura 5.12: Comparando variáveis de instância

Para a ação desse evento, selecione o `inimigo1` novamente e escolha a ação `Simulate Control`. Como escolhemos o comportamento de plataforma para ele, conseguimos simular o seu movimento com essa ação. Como o nosso jogo acontece da esquerda para a direita, faremos com o inimigo se mova para a esquerda.

Para isso, na janela que abrir, coloque o valor `Left` (esquerda) em `Control` (controle):

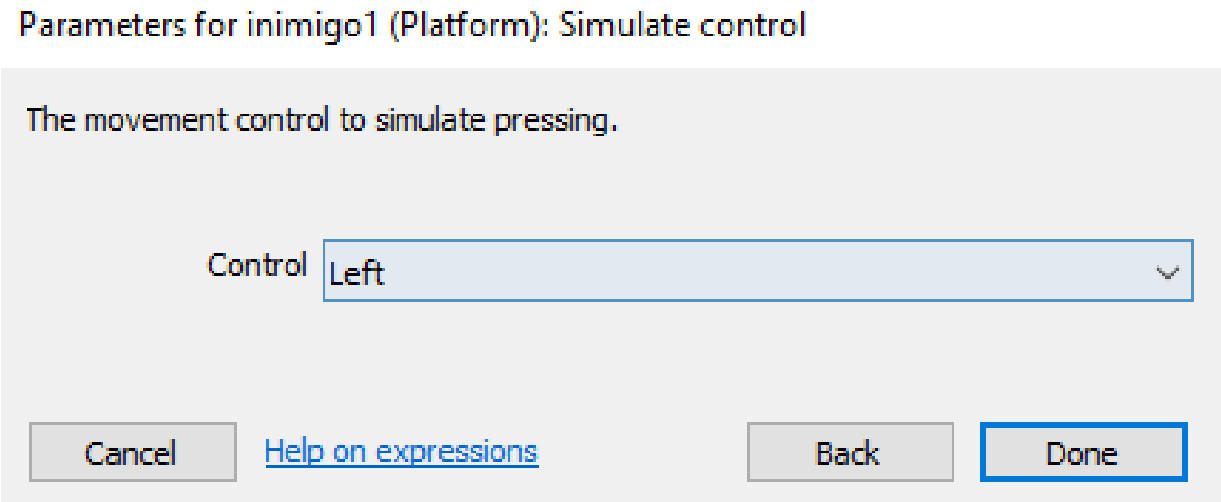


Figura 5.13: Simulate control

Então, se o `inimigo1` estiver se movendo, ele simulará o movimento de plataforma para a esquerda. Vamos adicionar outro evento para sempre setar a variável local `movendo` para `um` se o `inimigo1` estiver se movendo, como explicamos anteriormente.

Para isso, selecione novamente o `inimigo1` na janela `Add event` e adicione a condição `Is on-screen`. Para a ação desse evento, selecione o `inimigo1` e, em `Set value`, coloque o valor de sua variável local `movendo` com o valor `1`.

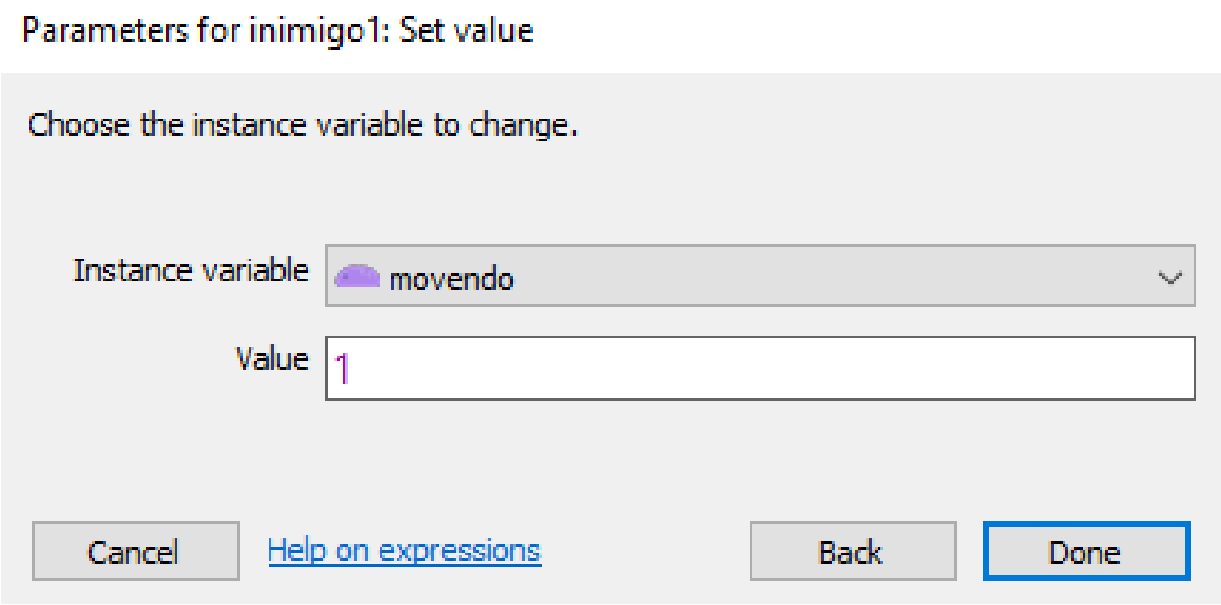


Figura 5.14: Set value

Pronto, os eventos correspondentes ao nosso `inimigo1` já estão conluídos. Ao final, sua folha de eventos deve estar assim:

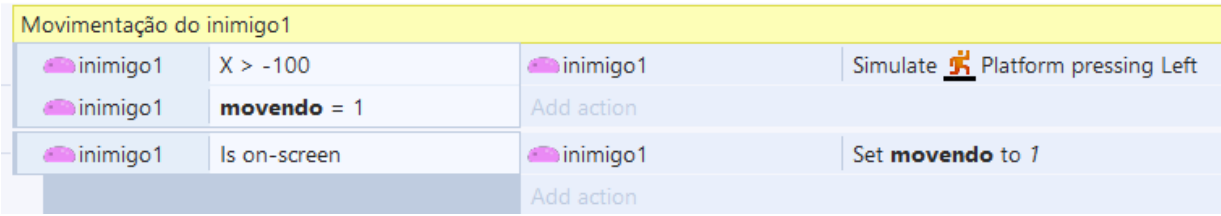


Figura 5.15: Movimentação do inimigo 1

Programando o inimigo2

Vamos partir para o `inimigo2`. Para este, usaremos três eventos diferentes.

1. O primeiro será responsável por fazer o inimigo pular a cada X segundos;
2. O segundo vai controlar o quadro de animação para quando ele estiver pulando;
3. O terceiro vai fazer o contrário do segundo.

Para realizarmos o primeiro evento, selecione o objeto `System` (objeto de sistema) na janela `Add event`. Escolha a condição chamada `Every X seconds` (a cada X segundos). Como o próprio nome diz, conseguimos definir ações para que se repitam a cada ciclo de X segundos.

Para o nosso jogo, inicialmente usaremos 1,5 segundos. Para configurar este número, coloque-o no parâmetro `Interval` (intervalo), conforme mostra a figura:

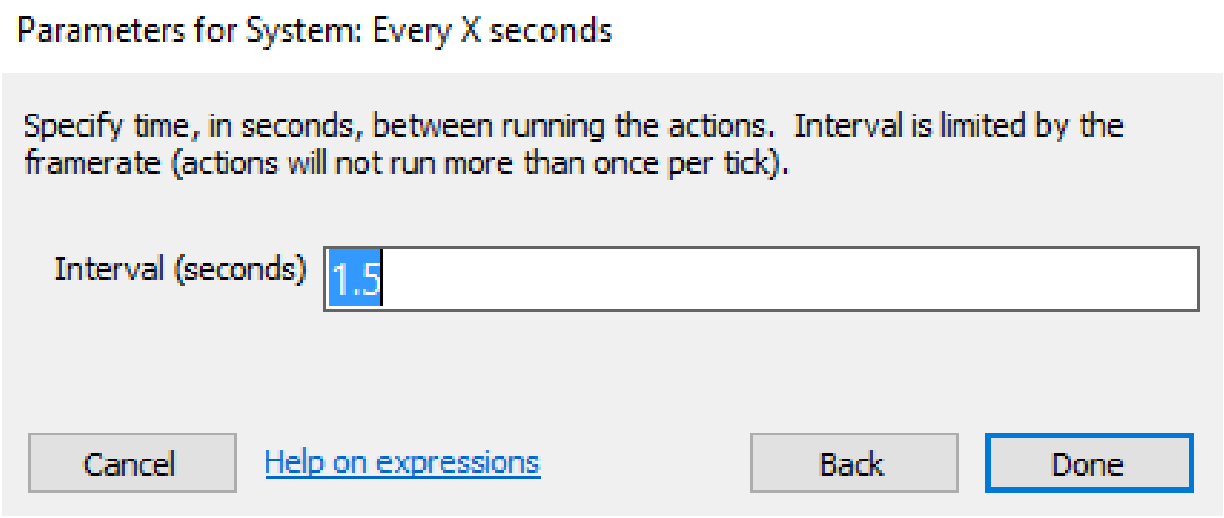

Parameters for System: Every X seconds
Specify time, in seconds, between running the actions. Interval is limited by the framerate (actions will not run more than once per tick).
Interval (seconds)
1.5
Cancel
Help on expressions
Back
Done

Figura 5.16: Every X seconds

Na ação deste evento, selecione o `inimigo2` e escolha a ação `Simulate Control`. Lembra-se de que usamos esta mesma ação no primeiro inimigo? Agora vamos configurá-la de forma diferente. Na janela que abrir, coloque o valor `Jump` (pulo) no campo `Control`. Isso fará com que o `inimigo2` pule. Muito bem, configuramos o primeiro evento da nossa lista.

Para o segundo evento, selecione novamente o `inimigo2` e selecione a condição `Is jumping` (está pulando). Ela verifica se o objeto em questão está pulando ou não. Vamos capturar esse pulo do inimigo para trocar o seu quadro de animação. Para fazer isso, na ação desta condição, selecione novamente o `inimigo2` e vá à opção `Set frame`. No campo `Frame number`, coloque o valor 0.

Lembre-se de que o zero representa o primeiro quadro da animação do personagem.

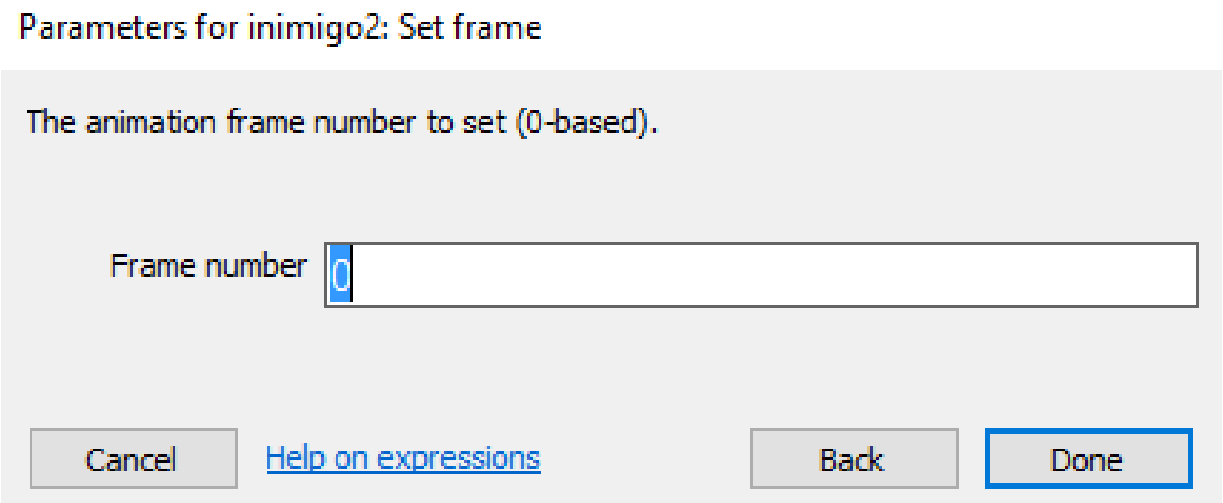


Figura 5.17: Setando o valor do frame

Feito isso, temos de partir para o terceiro item da lista, o evento contrário ao que acabamos de criar – ou seja, mudar o quadro de animação para quando o inimigo não estiver mais pulando. Selecione o objeto `System` e a condição `Else` para ele.

A condição `Else` sempre representa o caminho alternativo de uma dada condição. Como fizemos uma condição para quando o inimigo pular, conseguimos usar o `Else` para dizer o que o jogo deve fazer quando ele não estiver pulando.

Na ação selecione o `inimigo2` , vá em `Set frame` e, no campo `Frame number` , coloque o valor 1. Os eventos da movimentação do `inimigo2` devem estar assim na sua folha de eventos:

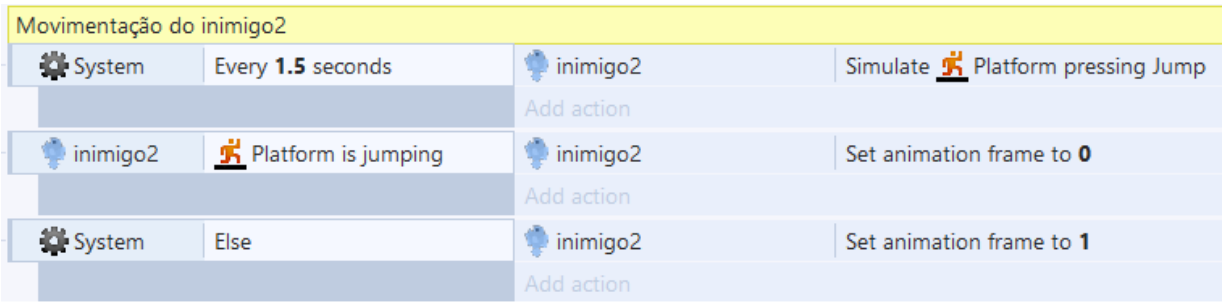


Figura 5.18: Movimentação do inimigo 2

Tudo certo? Perfeito. Vamos ao `inimigo3`.

Programando o Inimigo3

Como o `inimigo3` já tem o comportamento de `Sine` inserido, não precisamos adicionar os seus movimentos aqui. Já os configuramos nas propriedades do próprio comportamento anteriormente.

5.3 Jogador versus Inimigos

Fizemos um excelente trabalho configurando a "inteligência" dos inimigos dentro do jogo. Mas você deve ter notado que, até então, eles são inofensivos. Por que será?

Acontece que ainda não configuramos as colisões entre o nosso jogador e os seus inimigos. Ou seja, por enquanto, eles são indiferentes uns aos outros. Nesta seção, vamos alterar isto.

Jogador versus Inimigo1

O `inimigo1` será o único que o nosso personagem vai conseguir "destruir" no jogo. Faremos com que, caso o jogador consiga atingi-lo em cima de sua cabeça, ele morrerá. Este comportamento é muito comum em jogos de plataformas. Com toda a certeza, vieram

à sua mente jogos como Mario, Sonic e Crash, em que alguns inimigos são atingidos quando pulamos em suas cabeças.

Escolha o objeto `Jogador` na janela de adição de eventos (`Add event`) e escolha a opção `On collision with another object` (na colisão com outro objeto). Este evento é disparado quando um objeto entra em colisão com outro.

Vamos configurar inicialmente o que acontece quando o nosso jogador colide com o `inimigo1` . Quando isso acontecer, queremos que o nosso personagem perca uma das vidas e pisque por alguns segundos, indicando para quem está jogando que ele acabou de levar dano. Caso ele acerte a cabeça do inimigo, o jogador sai invicto.

Iniciaremos configurando o evento para quando acertarmos a cabeça do inimigo. Para isso, na janela que abrir, no campo `Object` , clique sobre `<click to choose>` , como mostra a figura:

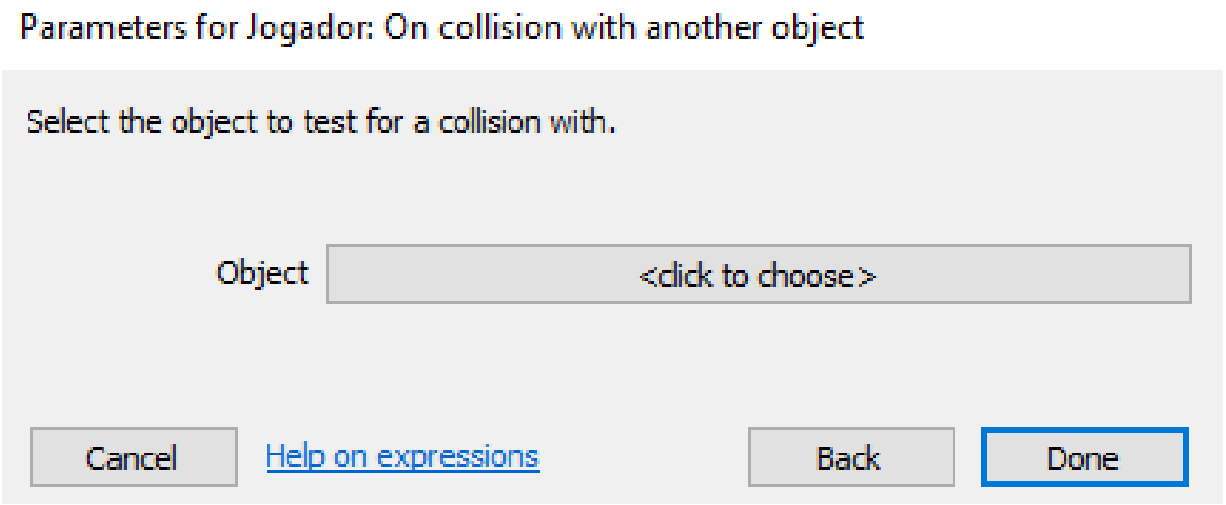


Figura 5.19: On collision with another object

Escolha o nosso `inimigo1` , como vemos a seguir:

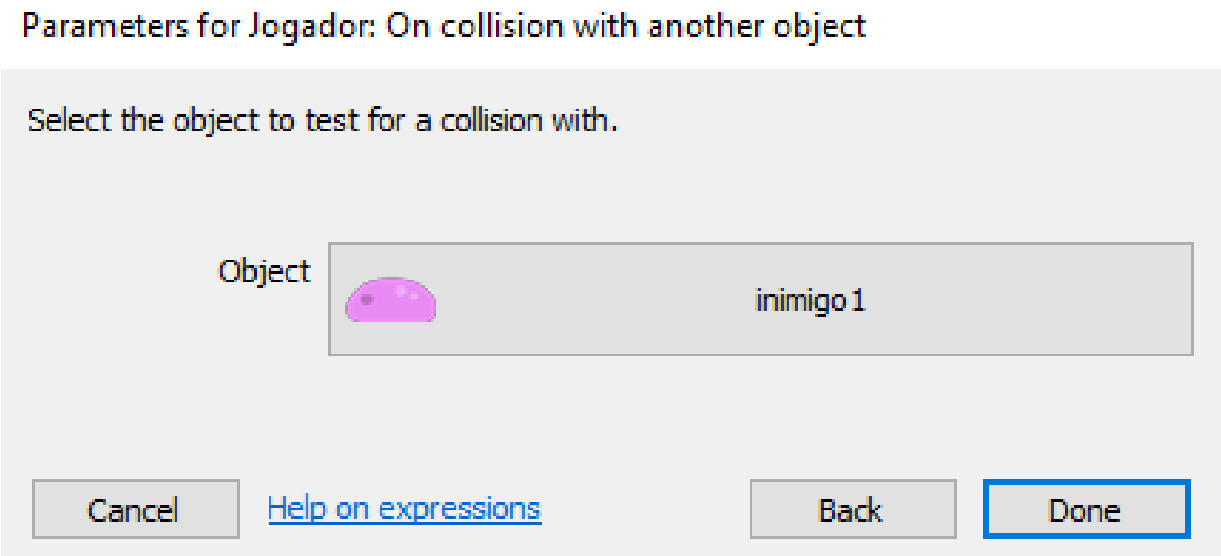


Figura 5.20: Escolhendo o objeto da colisão

Agora, vamos colocar dois subeventos nesse evento para quando o valor do Y do `Jogador` for menor que o valor Y do `inimigo1` . Isto é, quando o `Jogador` colidir por cima do `inimigo1` , este será destruído; caso contrário, se o `Jogador` estiver piscando (`Is flashing`), ele vai perder uma vida e setar o quadro de animação da `Sprite coracoes` .

Vale lembrar como criamos um subevento: clique com o botão direito do mouse no canto esquerdo da condição inserida e selecione a opção `Add` . Em seguida, clique em `Add sub-event` .

Para o primeiro subevento, queremos comparar se o valor da coordenada Y do `Jogador` é menor que a coordenada Y do `inimigo1` . Para isso, selecione o objeto `Jogador` e a condição `Compare Y` . Adicione os seguintes parâmetros para fazer a verificação do valor de Y:

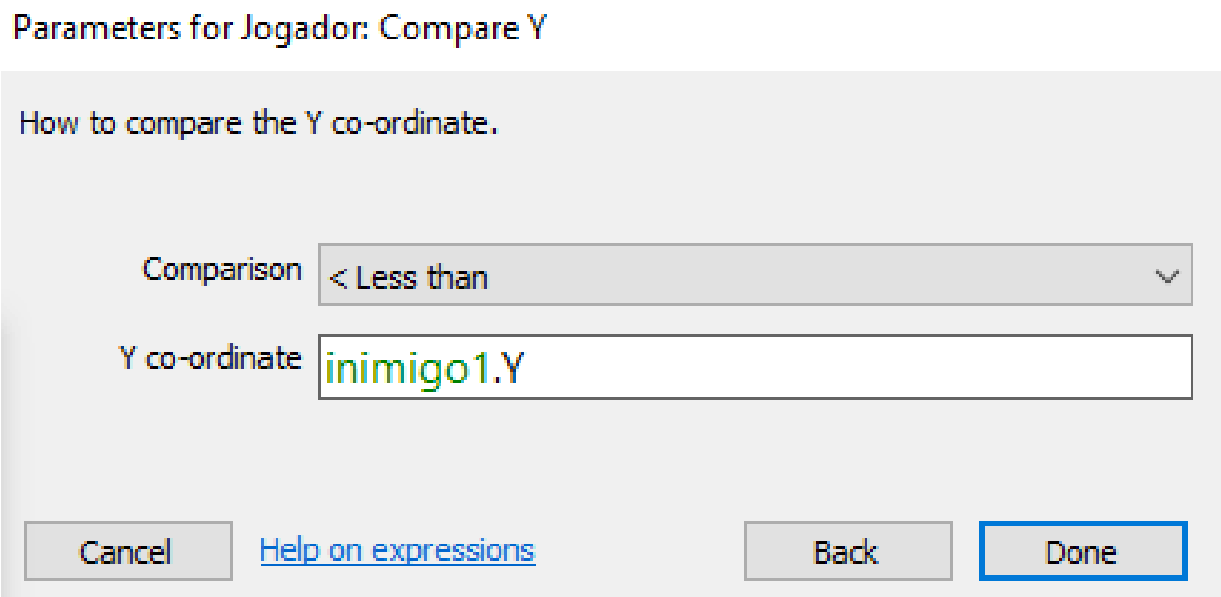


Figura 5.21: Comparando o valor de Y

Ótimo. Agora, para que a animação do personagem não fique estranha ao pular na cabeça do adversário, faremos com que ele dê um leve pulo para cima ao atingir o alvo. Conseguimos fazer isso usando a ação `Set vector Y`. Ela é responsável por configurar a posição do objeto no eixo Y.

Vamos escolher o objeto `Jogador` e a ação `Set vector Y`. Então, coloque o valor `-700` no campo `Vector Y`:

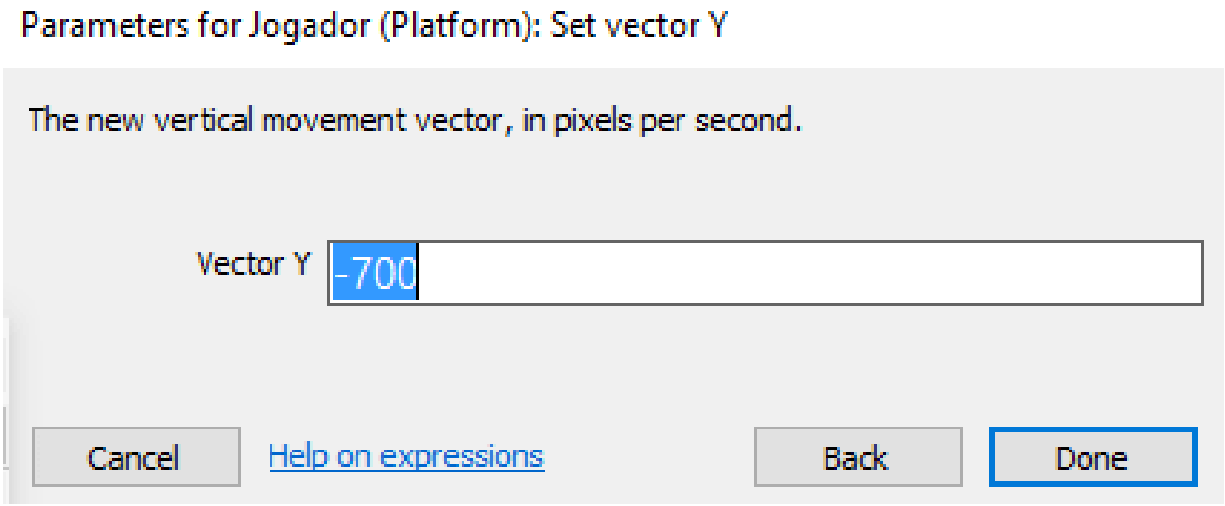


Figura 5.22: Set vector Y

Talvez você se pergunte por que estamos usando um valor negativo no eixo Y, sendo que queremos fazer com que ele pule. Fazemos isso porque o ponto de origem do plano cartesiano que define a posição dos objetos dentro do jogo está no canto superior esquerdo do seu monitor. Logo, para que o jogador vá para cima, temos de subir negativamente o eixo Y.

No nosso caso, configuramos 700 pixels. Se usarmos um valor positivo, em vez de subir, ele acabará descendo. Engraçado, né? Parece confuso a princípio, mas é só lembrar que o ponto de referência está no canto superior esquerdo do seu monitor.

Falta adicionar mais uma ação nesse evento. Nosso inimigo não pode ficar na tela depois de o destruirmos. Para removê-lo, usaremos a ação `Destroy` (destruir). Dentro dela, selecione agora o `inimigo1` e dê a ação de `Destroy`. Isso fará com o que o `inimigo1` seja destruído do jogo.

Perfeito, configuramos como eliminar o inimigo. Agora falta o segundo subevento, no qual configuraremos o que acontece quando o personagem toca o inimigo em qualquer outro lugar. Selecione o objeto de sistema (`System`) na janela `Add event`. Escolha a condição

`Else` e, no mesmo evento, coloque mais uma condição, como já fizemos anteriormente.

Clique com o botão direito do mouse no canto esquerdo da condição inserida. Selecione a opção `Add` e, depois, em `Add another condition` (ou clique em `C` do seu teclado). Selecione o `JogadorAnimacao` e a condição `Is flashing`. No evento criado, clique com o botão direito do mouse em cima do `Is flashing` e selecione a opção `Invert` (inverter); isso fará com que a ação seja invertida.

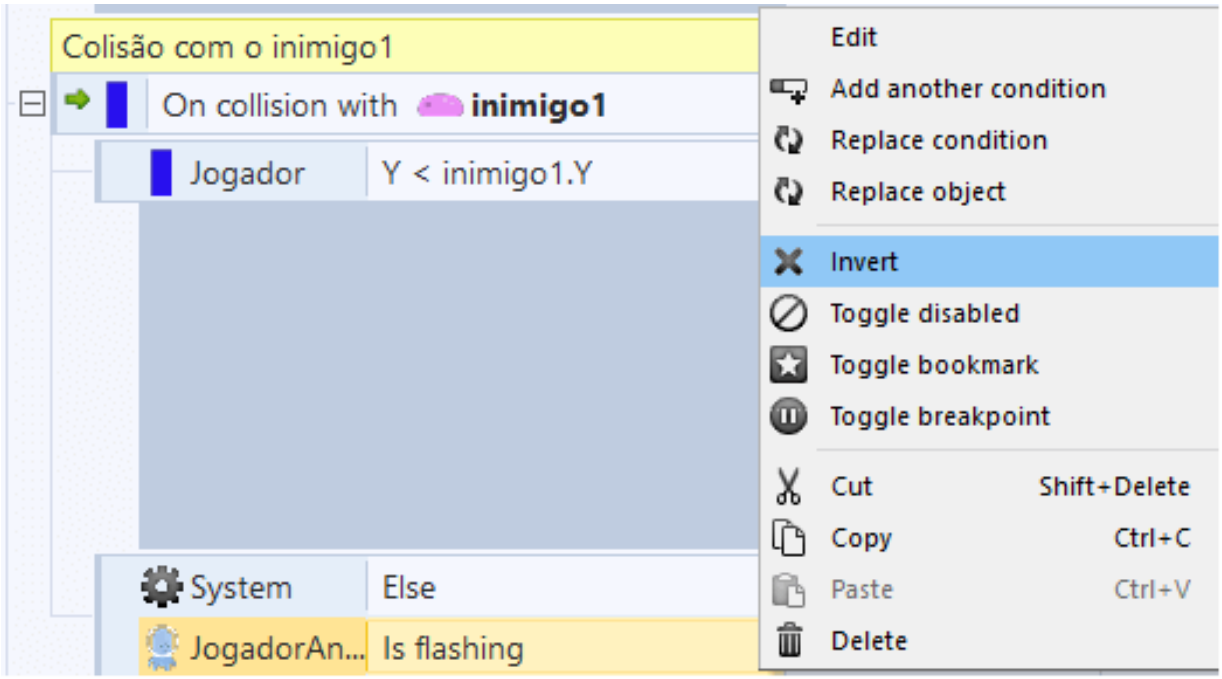


Figura 5.23: Is flashing

Só queremos que o jogador perca uma vida quando ele não estiver piscando. Para a ação desse evento, selecione `JogadorAnimacao` e dê a ação `Subtract from` (subtrair de). Isso fará com que seja subtraído um determinado valor – no nosso caso, queremos subtrair `1` da variável `vida` do nosso personagem. Para isso, coloque os parâmetros seguintes na janela que abrir:

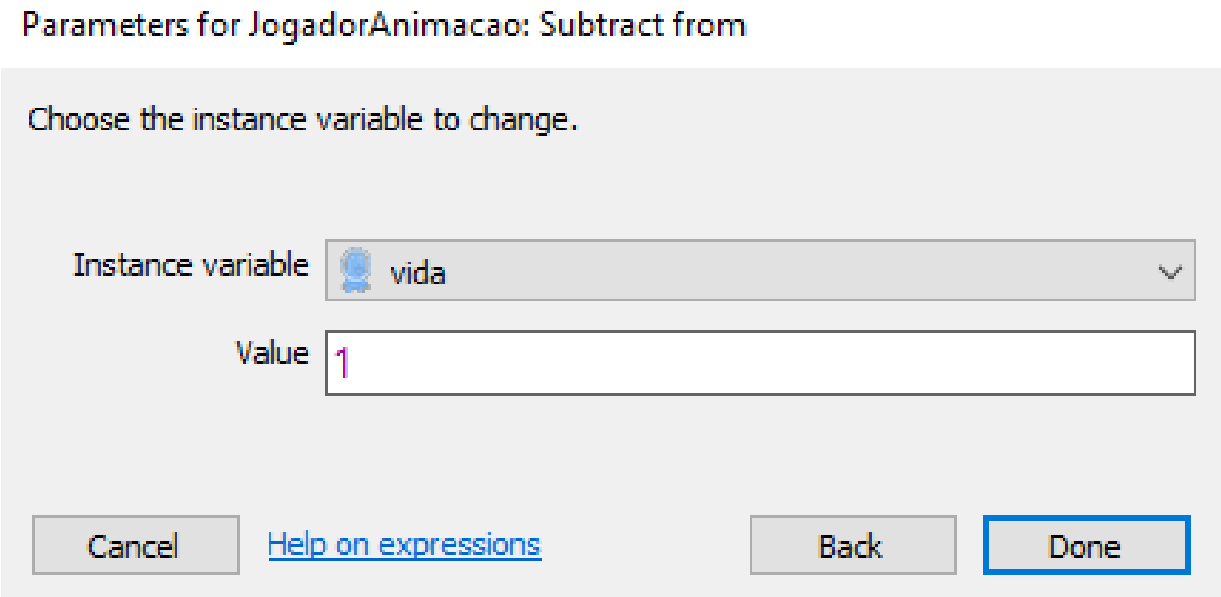


Figura 5.24: Subtract from

Na outra ação, selecione novamente o `JogadorAnimacao` e a ação `Flash`, para fazer a animação do jogador piscar. Não mude os parâmetros da janela que vai abrir em seguida, pois queremos exatamente esse valor de duração e tempo para o comportamento de piscar. Então, clique em `Done`.

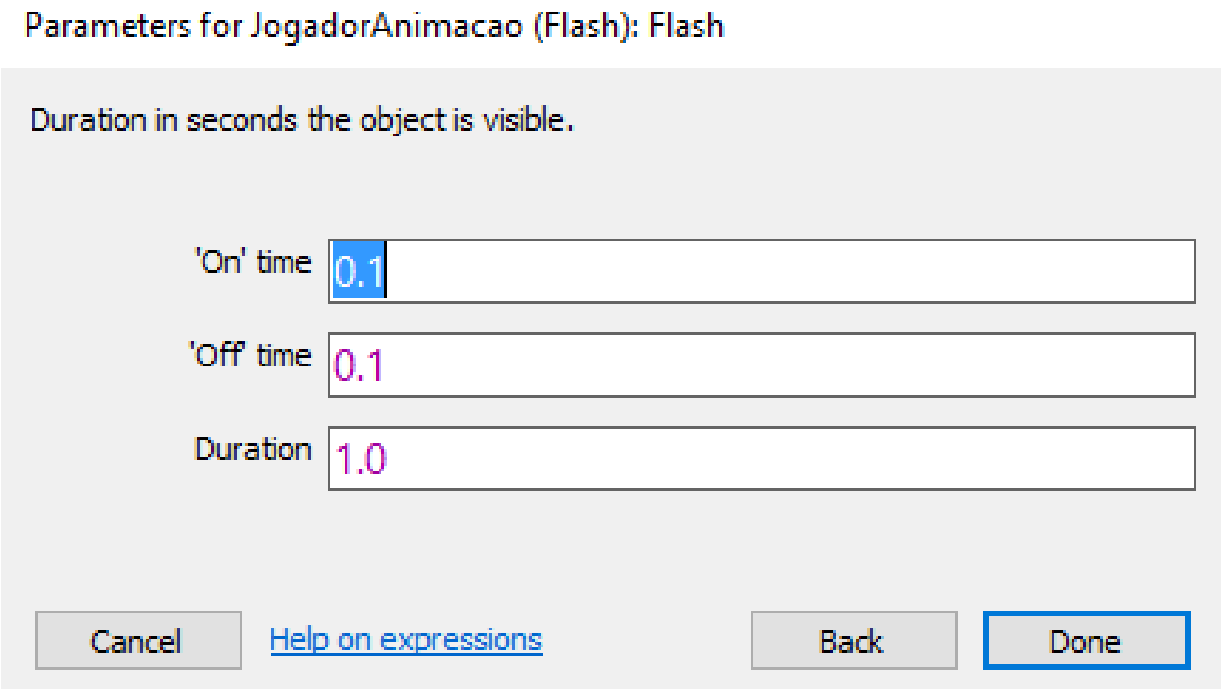


Figura 5.25: Flash

Os eventos de colisão do `inimigo1` deverão ficar igual aos da figura a seguir:

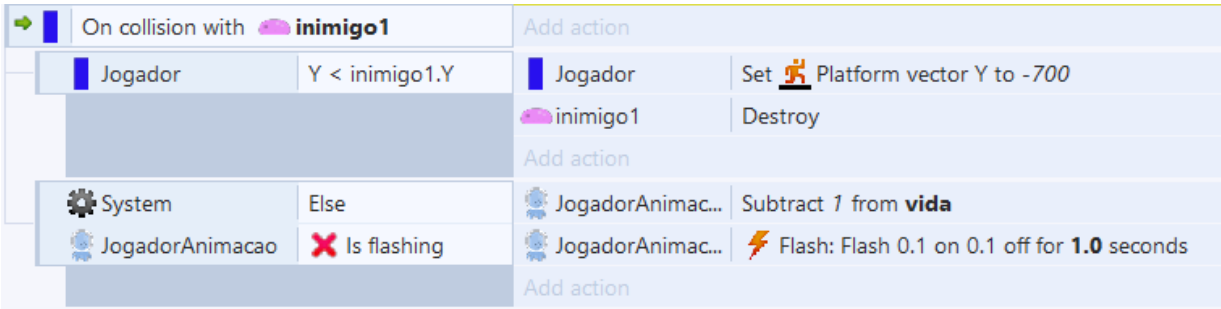


Figura 5.26: Jogador versus Inimigo1

Agora, se tocarmos no inimigo, perdemos uma vida, e o nosso jogador fica piscando por um breve período. Quando ele estiver piscando, ele fica invulnerável por um tempo, dando um espaço para o jogador se recompor. Depois deste pequeno tempo, ele fica vulnerável a danos de novo.

Para os próximos inimigos, nosso personagem não será capaz de atingi-los.

Jogador versus Inimigo2

Para o segundo inimigo, vamos criar somente um evento para, quando o `Jogador` não estiver piscando, perder uma vida ao colidir com ele. O processo é muito semelhante ao que já fizemos, então, será fácil.

Para isso, escolha o objeto `Jogador` na janela de adição de eventos (`Add event`) e escolha a opção `On collision with another object` . Na janela que abrir, escolha o `inimigo2` no campo `Object` :

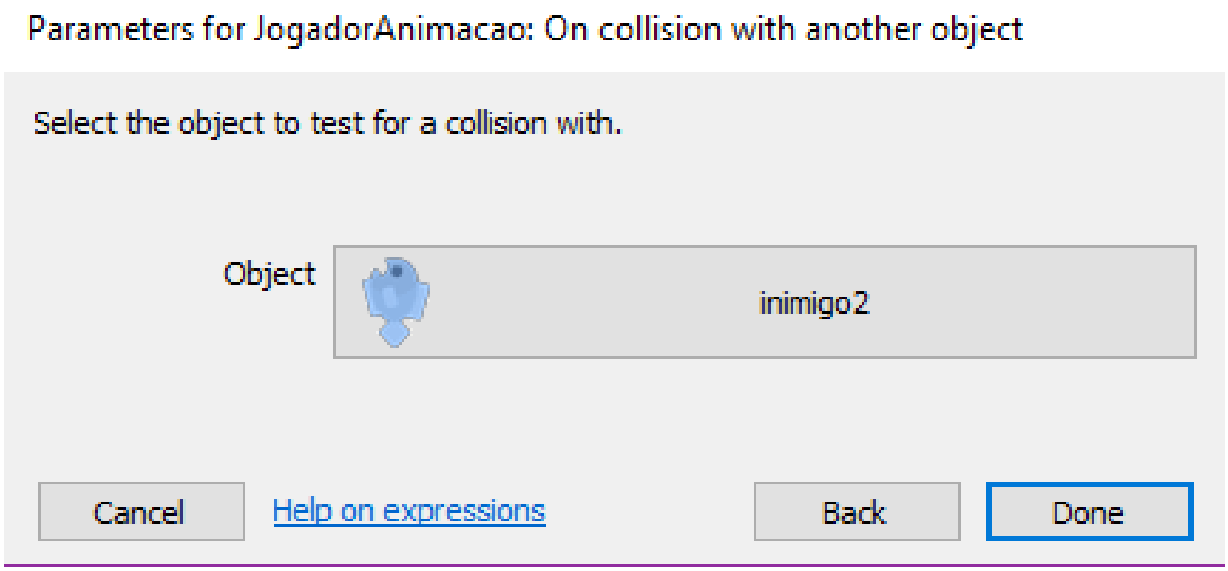


Figura 5.27: On collision with another object

Coloque uma segunda condição nesse evento e selecione o `JogadorAnimacao` e a condição `Is flashing` . No evento criado, clique com o botão direito do mouse em cima do `Is flashing` e selecione a opção `Invert` , portanto, ele só perderá vida se não estiver piscando.

Na ação desse evento, coloque as mesmas ações que colocamos no `inimigo1` para subtrair uma vida e fazer o nosso personagem "piscar". Relembre que, para subtrair um ponto de vida, selecione

`JogadorAnimacao` , dê a ação `Subtract from` e escolha `um` no campo `value` . E para fazer o jogador piscar, selecione o `JogadorAnimacao` e a ação `Flash` e, então, clique em `Done` .

Você terá o seguinte evento para o `inimigo2` :

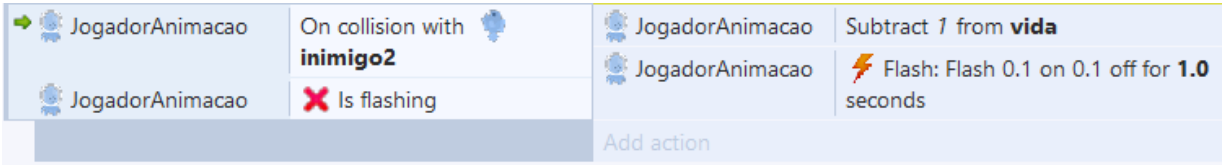


Figura 5.28: Jogador versus Inimigo2

Jogador versus Inimigo3

Para o `inimigo3` vamos fazer o mesmo evento anterior. A única diferença aqui é que colocaremos o `inimigo3` no campo `Object` , na janela `On collision with another object` , pois agora vamos programar as ações do `inimigo3`

Seu evento ficará assim:

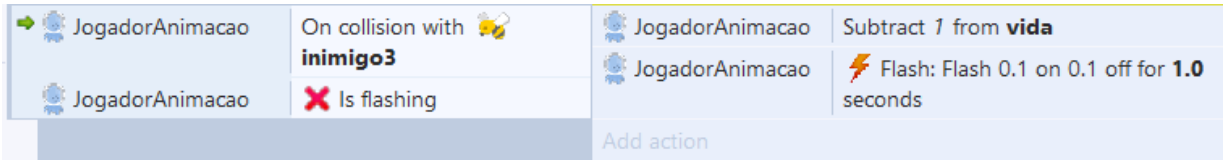


Figura 5.29: Jogador versus Inimigo3

Maravilha. Configuramos todos os comportamentos adequados para os inimigos, desde suas animações, movimentações e até o que deve acontecer quando o personagem tocar neles. Fique à vontade para alterar os valores que sugerimos e brinque com o que fizemos.

Nos próximos capítulos, trabalharemos com um sistema de pontuação e de fim de jogo, para quando o personagem ficar sem vidas.

Capítulo 6
Telas de pausa e de fim de jogo

Uma característica comum em todos os jogos – independente de sua categoria, estilo ou plataforma – é que eles possuem uma maneira do jogador perder. Se for um jogo de corrida, ele perde se não conseguir chegar na primeira colocação. Se for de futebol, ele perde se não vencer o campeonato. Se for de luta, se não vencer os seus adversários na porrada.

Mas e em um jogo de plataforma? Podemos nos basear nos grandes sucessos do mercado para nos trazer inspirações nos nossos jogos. Assim como no *Super Mario Bros.* – talvez a melhor referência que temos quando falamos em jogos de plataforma 2D –, fizemos com que o nosso personagem perca vidas ao entrar em contato com os inimigos. Entretanto, ainda não programamos o que acontece quando o jogador perde todas elas.

Neste capítulo, criaremos a tela de final de jogo (*game over*) do nosso jogo. Ela aparecerá toda vez que as vidas do jogador se esgotarem. E para aproveitar o embalo, também vamos desenvolver uma tela de pausa, afinal, não há nada pior do que não poder pausar o jogo para um intervalo emergencial, não é mesmo?

6.1 Tela do final do jogo

Vamos criar agora a tela de *game over*, e programar para ela aparecer quando o personagem ficar sem vidas. Faremos a tela com dois botões:

1. Jogar novamente
2. Voltar para a página inicial

O primeiro botão dará ao jogador a opção de voltar à fase que ele estava jogando, ou seja, dar uma nova tentativa para que ele consiga superá-la. Para o outro botão, já deixaremos as coisas preparadas para implementarmos o sistema de seleção de fases do jogo.

Ele funcionará de forma muito semelhante ao dos jogos da franquia *Angry Birds*. Neste sistema, o jogador só consegue acessar a fase atual ou jogar novamente as anteriores. As fases posteriores ficam travadas até que ele consiga vencer a atual.

Mas faremos tudo isso aos poucos. Neste primeiro passo, criaremos somente a tela. Esta, na verdade, será uma caixa com as duas opções que vai descer no layout quando as vidas forem todas para o espaço. Para criá-la, insira um objeto do tipo `Sprite`, no layout `Fase1` com a layer `UI` selecionada, e nomeie-o de `Tela_gameover`. Importe neste objeto a imagem `tela-gameover`. Ela está na nossa pasta de recursos, dentro do diretório `Telas`.

Como faremos a tela deslizar para dentro do jogo quando necessário (momento em que o jogador perde todas as vidas), por ora, precisamos colocá-la fora do layout. Portanto, coloque a `Sprite` `tela-gameover` fora do layout (assim como fizemos com o objeto `JogadorAnimacao`), conforme a seguir:

Figura 6.1: Tela de game over acima do layout

Usando a folha de eventos, definiremos sua posição corretamente quando for preciso. Para configurar isso, entre nessa folha e crie um novo evento. Ao criar, selecione o objeto `JogadorAnimacao` . Como queremos que a tela apareça sempre que o jogador não tenha mais vidas, temos de saber quando isso acontece.

Você se lembra daquela variável de instância, chamada `vidas` , que criamos no capítulo anterior? Ela será essencial agora. Utilizaremos um evento de comparação para saber quando ela estiver com o valor zero. Para isso, no evento criado, escolha a condição `Compare instance variable` (comparar variável de instância).

Na janela seguinte, teremos algumas opções. No campo `Instance variable` (variável de instância), selecione a variável que queremos

validar – neste caso, a `vidas`. No segundo campo, temos o `Comparison` (comparação), no qual vemos vários tipos de comparação:

- `=` – *Equal to* (igual a)
- `?` – *Not Equal to* (não é igual a)
- `<` – *Less than* (menor que)
- `=` – *Less or equal* (menor ou igual)
- `\>` – *Greater than* (maior que)
- `=` – *Greater or equal* (maior ou igual)

Ao ler isso, você provavelmente deve ter se lembrado das suas aulas de matemática, aquelas que ensinavam as funções geométricas com comparações de valores, não é mesmo? A lógica aqui é a mesma.

Podemos utilizar essas comparações para fazer inúmeros tipos de lógicas em nossos jogos. Para este caso, queremos somente saber quando o valor é zero, então escolha a opção `Equal to`. O último campo, chamado de `Value` (valor), é onde colocamos o valor que desejamos comparar. Insira o valor `0` (zero).

Ao final, sua janela deve estar preenchida assim:

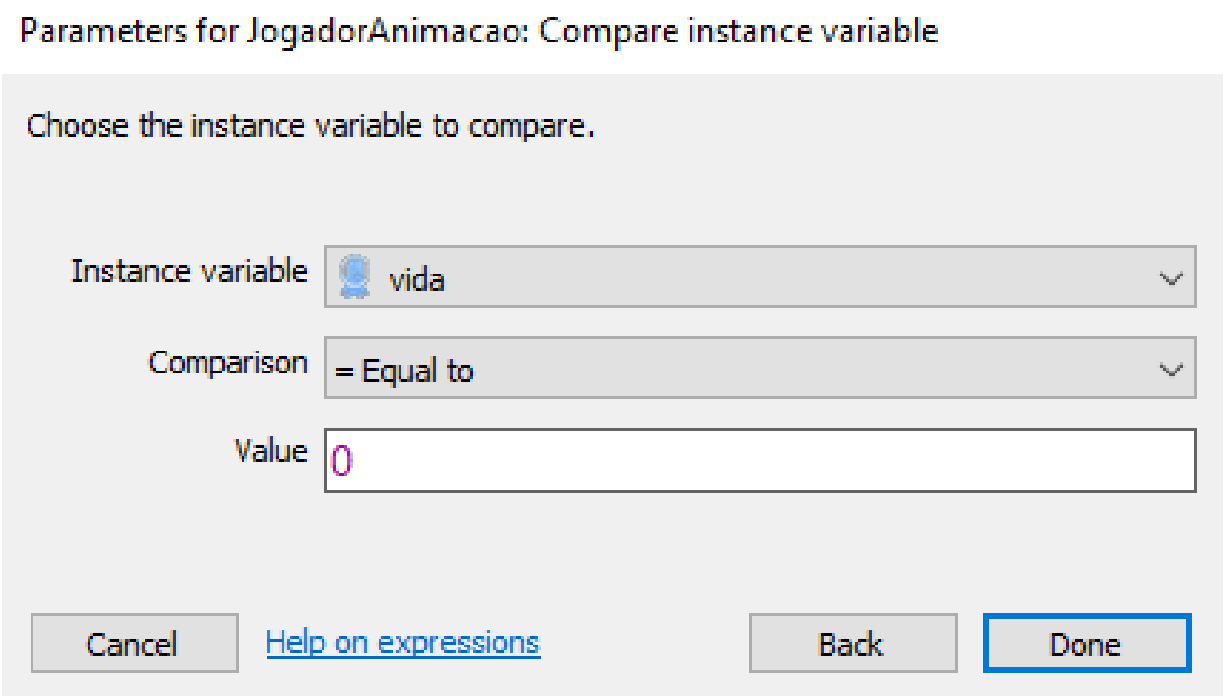


Figura 6.2: Compare instance variable

Agora que sabemos quando o jogador zera suas vidas, temos de associar as ações que queremos. Colocaremos três ações para esse evento.

Na primeira ação, vamos definir a posição que a tela de final de jogo vai aparecer no layout. Na segunda, removeremos o nosso personagem do jogo para ele não continuar se movendo depois dele ter sido finalizado. E na terceira e última ação, vamos congelar o restante do cenário.

Agora que já sabemos quais serão as funções de cada uma das ações que serão inseridas, vamos começar a programá-las. Para a primeira, selecione o objeto `telaGameover` e escolha a condição `Set position` (setar posição). Lembre-se de que usamos essa função sempre que queremos definir a posição de um objeto no layout com suas coordenadas cartesianas. Coloque 320 para o X, e 240 para o Y (esta posição corresponde ao meio da nossa janela).

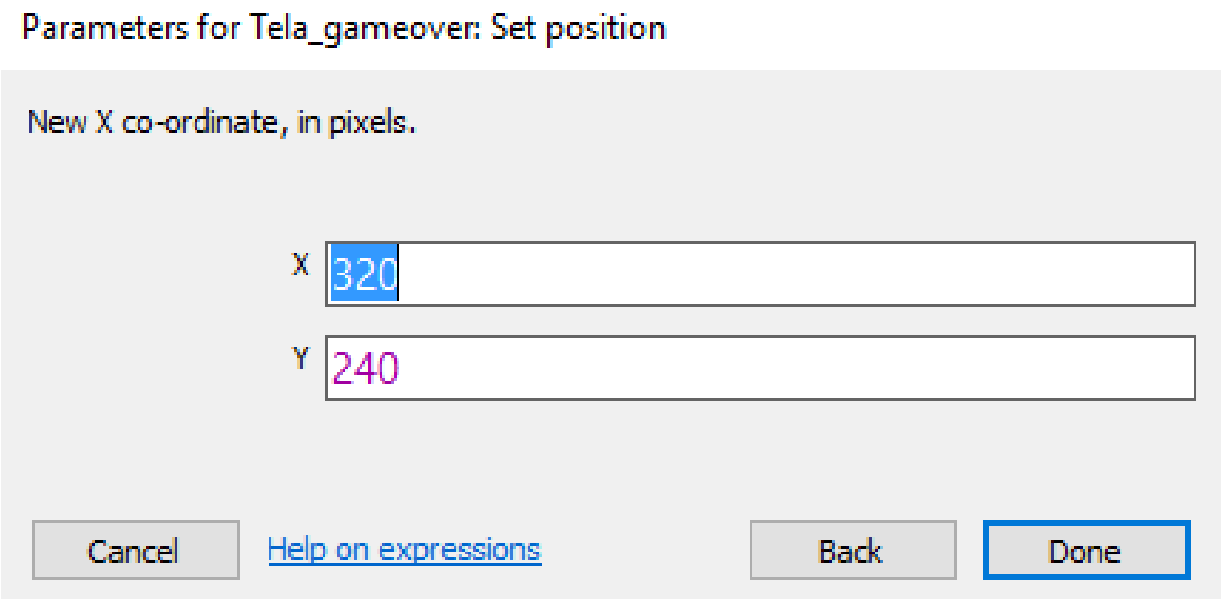


Figura 6.3: Set position

Para a segunda ação, selecione o objeto `Jogador` e simplesmente escolha a ação `Destroy`. Assim como fizemos com os inimigos no capítulo anterior, essa função removerá o objeto do layout.

Para a terceira e última ação necessária, selecione o objeto de sistema (`System`) e a ação `Set time scale` (setar escala de tempo). Com essa função, conseguimos alterar a escala de tempo do jogo no Construct 2. Em outras palavras, é possível alterar a velocidade com que o processamento das coisas ocorre dentro do cenário.

Por padrão, tudo no Construct 2 acontece na escala `1` (um). Se aumentamos a escala para `2` (dois), a velocidade dobra. Nesta mesma lógica, se reduzirmos para zero... Acontece isso mesmo o que você pensou, tudo fica "congelado". Coloque o valor `0` no campo `Time scale`, como a seguir:

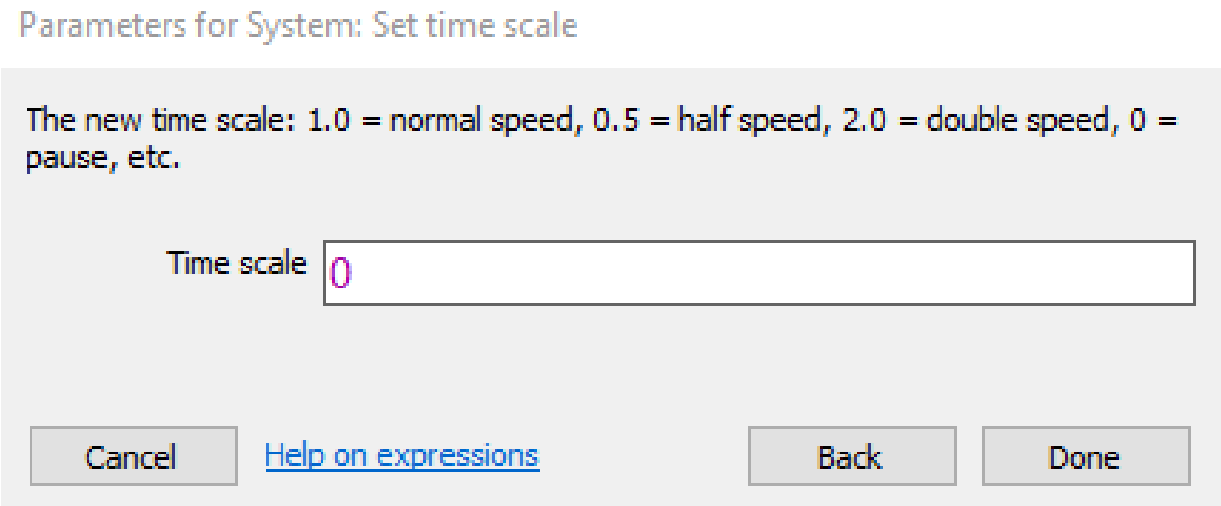

Parameters for System: Set time scale
The new time scale: 1.0 = normal speed, 0.5 = half speed, 2.0 = double speed, 0 = pause, etc.
Time scale
0
Cancel
Help on expressions
Back
Done

Figura 6.4: Set time scale

Ao final, o evento deve estar configurado desta forma:

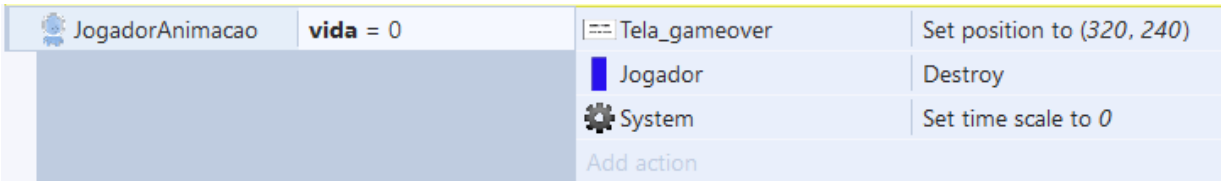


Figura 6.5: Evento da tela de game over

Muito bem! Configuramos a primeira parte da nossa tela de final de jogo. Repare que ainda não trabalhamos nos botões que farão o jogador voltar ao mapa ou à seleção de fases. Faremos isso muito em breve, no capítulo *Trabalhando com botões*. Lá explicaremos todos os conceitos necessários para que você consiga utilizar quantos botões quiser dentro do jogo.

Feito isso, retomaremos esta tela e faremos o restante de sua lógica. Por ora, vamos aproveitar o conceito de escala de tempo que aprendemos nesta seção e adicionar uma tela de *pause* no jogo.

6.2 Criando a tela para pausar

Antes de avançarmos com o desenvolvimento, vamos parar um instante para aproveitar o trem andando e adicionar uma funcionalidade que, apesar de não ser obrigatória para o funcionamento, tornará o jogo mais completo e agradável. Estamos falando na funcionalidade de pausa.

Quem nunca teve de pausar o jogo para atender um telefonema, responder uma mensagem ou ir beber algo? É importante que os jogos tenham esta funcionalidade (a não ser que este seja de multijogadores online, neste caso não faz sentido). Para fazer isso, vamos aproveitar um dos tópicos que acabamos de aprender: escala de tempo.

Como dissemos, a escala de tempo corresponde à velocidade com que o jogo acontece. Por padrão, ela começa em 1. Se dobramos este valor, por exemplo, tudo acontece duas vezes mais rápido. Da mesma maneira, se alteramos este valor para meio, tudo leva o dobro do tempo para acontecer. Logo, para fazer tudo congelar, usaremos o valor zero.

Assim como fizemos com a tela de fim de jogo, adicionaremos uma tela de pause que será exibida sempre que o jogo for pausado. Não faremos toda a funcionalidade redonda neste momento, mas vamos deixar tudo preparado para que, nos próximos tópicos, o desenvolvimento se torne mais fácil, ok?

Comece selecionando a layer `UI` do nosso layout da `Fase1`. Com ela selecionada, insira um objeto do tipo `Sprite` e nomeie-o de `Tela_pause`. Nele, importe a imagem `tela-pause` que está no diretório `Telas` da nossa boa e velha pasta de recursos. Coloque essa `Sprite` fora do layout, conforme fizemos com a tela de *game over*. Podemos colocar uma ao lado da outra, como a seguir:

Figura 6.6: Tela de pausa acima do layout

Não está faltando algo?

Neste capítulo, inserimos as telas de fim de jogo e de pausar. Aprendemos um pouco sobre o uso das variáveis de instância para realizar comparações e criar lógicas dentro da folha de eventos. Outro aspecto importantíssimo que assimilamos foi o de escala de tempo dentro do jogo.

Apesar de tudo, ainda não concluímos nossas telas. Em ambas, ficaram faltando os seus respectivos botões, certo? Para a de fim de jogo, teremos a opção de `jogar novamente` e `Voltar para a tela inicial`, enquanto na de pausa teremos a opção `Regressar ao jogo` ou `Voltar para a tela de seleção de fases`.

Será no layout da fase que incluiremos o botão responsável por fazer a tela deslizar para dentro dele e pausar o jogo. Mas não se preocupe, retomaremos o desenvolvimento dessas telas quando falarmos sobre os botões do Construct 2.

Capítulo 7
Inserindo itens coletáveis e pontuação

Neste capítulo, vamos trabalhar com outro elemento clássico de jogos de plataforma: os itens coletáveis. Estes nada mais são que objetos que o jogador tem a escolha de coletar ou não. Caso ele colete, é recompensado com pontos, vidas, invulnerabilidade ou até mesmo fases bônus; caso contrário, nada acontece. É somente uma maneira de recompensá-lo pelo seu esforço.

Os itens coletáveis podem ser qualquer coisa, tudo depende do contexto do seu jogo. No famoso jogo do *Super Mario Bros.*, lançado pela Nintendo, ele coleta moedas, flores, penas, cogumelos e estrelas, por exemplo. Cada um dos itens tem uma vantagem diferente. Já no jogo do ouriço *Sonic*, lançado pela Sega, os itens coletáveis são as argolas.

Cada um usa o universo de seu jogo para criar os seus itens. Para o nosso, não será diferente.

No decorrer deste capítulo, aprenderemos como criar itens colecionáveis e associá-los a um sistema de pontuação que será exibido na tela. Também faremos um sistema de chave e porta, que obrigará o jogador a procurar pela chave para poder vencer a fase. Além disso, usaremos nossos novos conhecimentos para conectar os pontos que trabalhamos nos capítulos anteriores.

7.1 Adicionando itens coletáveis

Nosso sistema de itens colecionáveis funcionará de forma extremamente simples. A princípio, criaremos três tipos:

- Moedas

- Estrelas
- Chave/Porta

As moedas e as estrelas funcionarão como itens de pontuação. A cada moeda que o jogador coletar, ganhará uma certa quantidade de pontos. As estrelas serão itens mais raros, mas que dão uma maior pontuação.

Por fim, vamos adicionar o sistema interessante de porta e chave. Ele também funcionará de forma bem simples: ao final de cada fase, colocaremos uma porta. O jogador só conseguirá abri-la e finalizar a fase se conseguir encontrar a chave que deixaremos pelo mapa.

Agora que esclarecemos nossos planos, vamos começar criando algumas moedas.

Inserindo as moedas

Para inserir as moedas no jogo, adicione uma nova `Sprite` na layer `Principal` e importe a imagem `moeda` da pasta `Itens`, que está dentro dos nossos recursos. Espalhe as moedas pela fase como desejar, porém, não as coloque acima de 290 pixel no eixo Y, pois este é o tamanho máximo que o pulo do jogador vai alcançar. Também tenha cuidado para não colocar mais de 30 moedas no cenário, pois o tamanho do layout é pequeno e, ao colocar muitas, o jogo começará a consumir muitos recursos de máquina, ok?

Para facilitar o trabalho de colocar várias moedas, podemos copiá-las. Para copiar, basta selecionar o objeto, clicar com o botão direito do mouse e selecionar a opção `Copy`, como mostra a figura seguinte (ou aperte o atalho `Ctrl + C` do seu teclado).

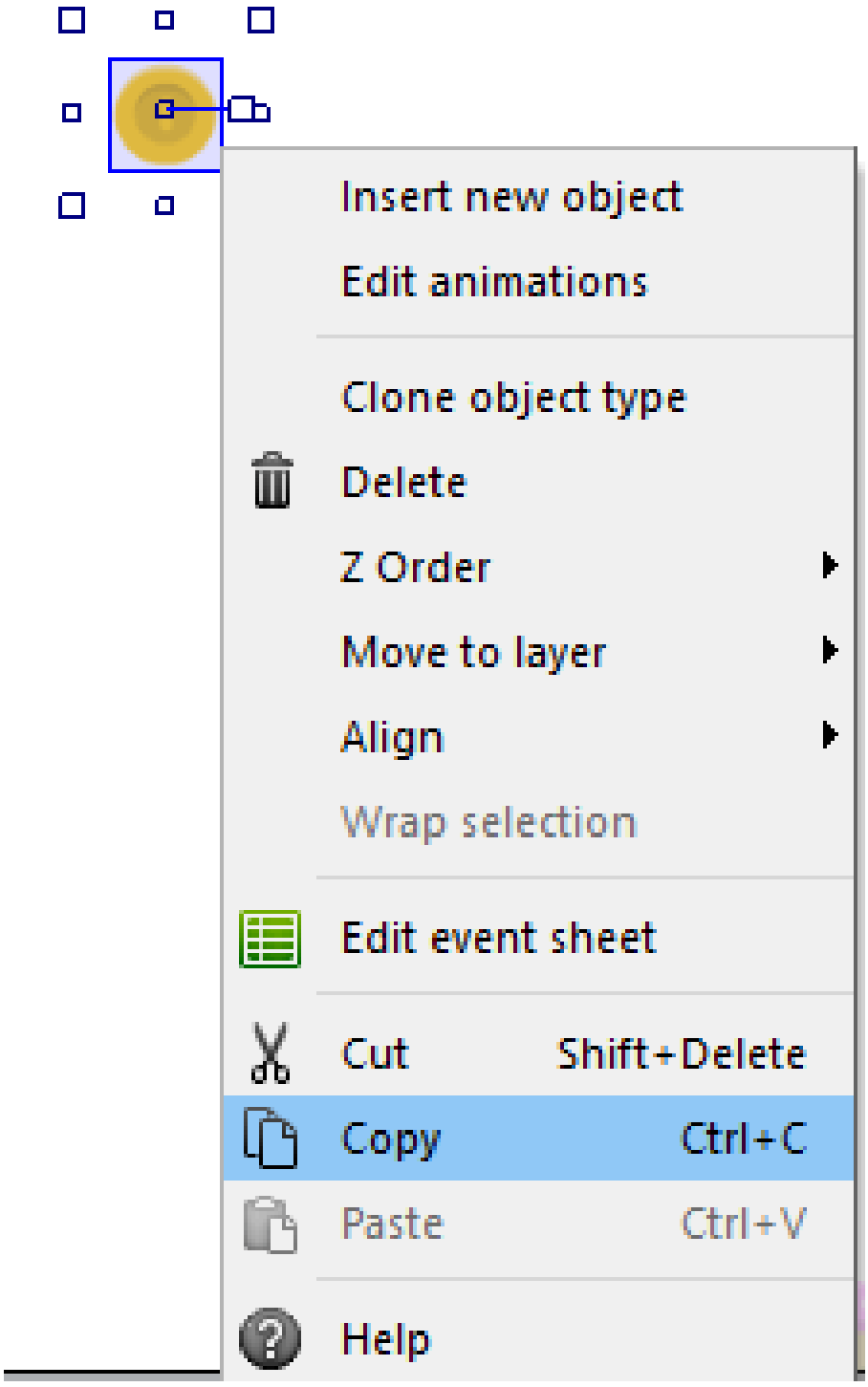


Figura 7.1: Copiando moedas

Depois, para colar, basta clicar na tela novamente com o botão direito do mouse na posição que desejar colá-lo e selecionar a opção `Paste` – ou apertar `Ctrl + V` do teclado.

Depois de inserir as moedas, precisamos adicionar um novo evento para quando o personagem colidir com elas. O primeiro passo é fazê-las serem removidas do cenário, para dar a impressão de que ela realmente foi coletada pelo jogador.

Para isso, abra a folha de eventos, crie um novo evento, selecione o objeto `Jogador` e a condição `On collision with another object` (em colisão com outro objeto). No campo `Object`, insira a `moeda` e, na ação, escolha a `moeda` e a ação `Destroy`.

O evento deve ficar assim:

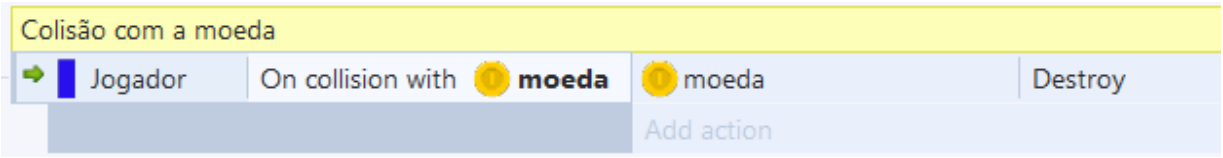


Figura 7.2: Colisão com a moeda

Isso fará com que a moeda desapareça do layout sempre que for coletada. Além de dar a impressão de que foi apanhada, ajudará a liberar memória da máquina. Feito isso, vamos adicionar algumas estrelas.

Inserindo as estrelas

A ideia é que as estrelas sejam itens mais raros do que as moedas e que ofereçam mais pontos como recompensa. Por isso, vamos nos limitar a colocar apenas três delas.

Para inseri-las, faremos o mesmo processo da inserção das moedas. Adicione uma nova `Sprite` na layer `Principal` e importe a imagem `estrela`, que está dentro dos recursos na pasta `Itens`.

Espalhe as três estrelas pela fase como desejar, porém, como já mencionamos, tome cuidado para não as colocar acima de 290 pixel no eixo Y, já que esse é o valor máximo que o jogador vai alcançar. Na folha de eventos, vamos adicionar um parecido com o de colisão

com as moedas; a única diferença é que agora selecionaremos o `Sprite estrela`, como mostra a figura:

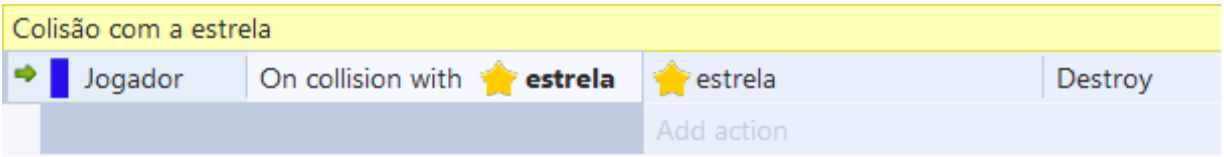


Figura 7.3: Colisão com a estrela

Muito bem, agora temos os nossos principais itens coletáveis espalhados pelo layout:

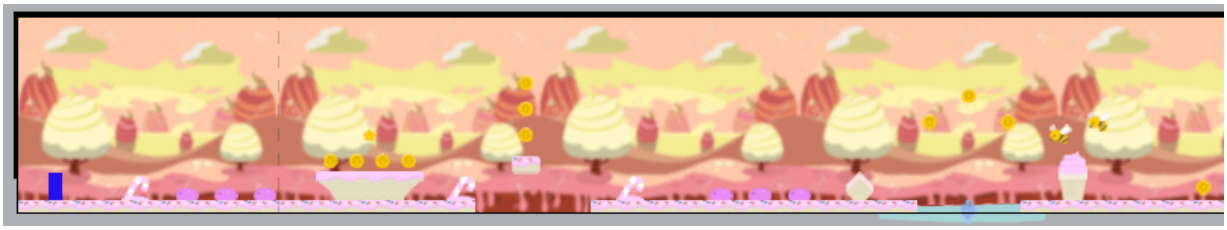


Figura 7.4: Itens espalhados pelo layout

Até o momento, os itens apenas são destruídos quando entram em contato com o jogador. Ainda neste capítulo, vamos adicionar uma pontuação para quando o jogador colidir com as moedas ou as estrelas. Por ora, continuaremos trabalhando com os itens coletáveis, e incluiremos a chave e a porta na fase.

Construindo o sistema de chave e porta

O objetivo do sistema de chave e porta é fazer com que o jogador explore melhor a fase. Muitos jogos utilizam essa estratégia, mesmo que não sejam necessariamente chaves e portas. A lógica por trás é bem simples.

Primeiro, precisamos inserir mais duas `Sprites` na layer `Principal`: uma corresponderá à porta do final da fase, e a outra à chave que vai abri-la. Para a porta, vamos adicionar uma `Sprite` com o nome `porta_fase1`.

Para representar os dois estados da porta (aberta e fechada), usaremos dois frames de animação – ambos estão na pasta de recursos, com o nome `porta1-fechada` e `porta1-aberta`. Importe as imagens, respectivamente, nos quadros 0 e 1.

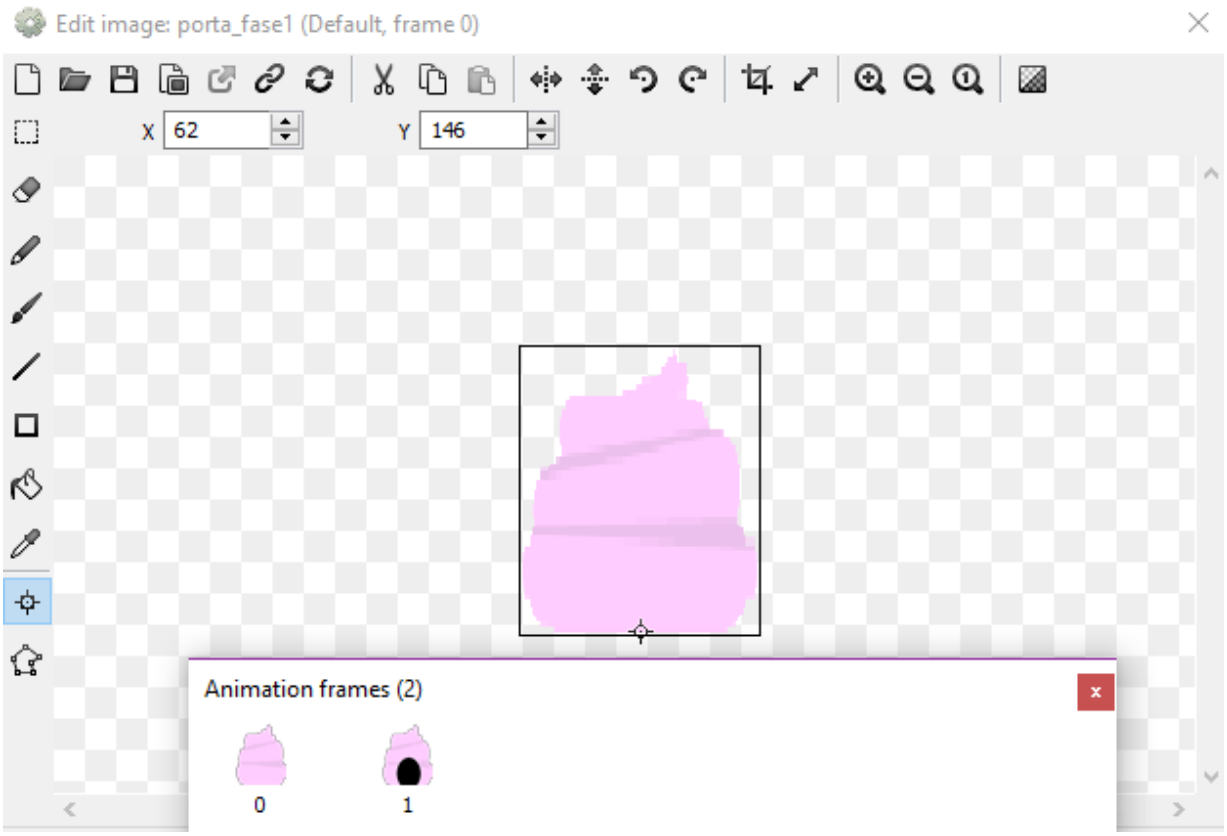


Figura 7.5: Frames da porta

Como já fizemos com os outros objetos para que não fossem animados automaticamente, vá até as propriedades de animação e coloque a velocidade (`speed`) para `0`, e configure a propriedade de repetição para `No`:

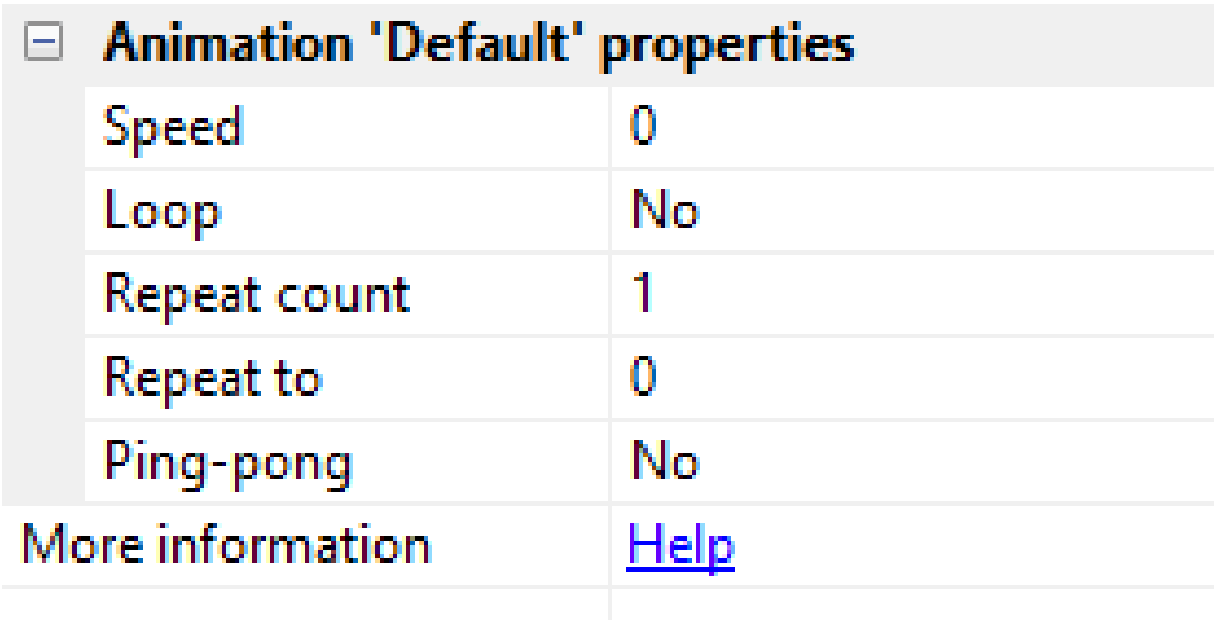


Figura 7.6: Animação da porta

Coloque a porta lá no final do seu layout. Agora, para a chave, vamos adicionar uma `Sprite` com o nome `chave`. Vamos importar nela a imagem de mesmo nome, contida na pasta `Itens` no diretório de recursos.

Coloque a chave em algum lugar do layout que o jogador consiga alcançar. Feito isso, precisamos programar o evento que associa a coleta da chave com a abertura da porta. Começaremos adicionando o evento para quando o jogador colidir com a chave.

Na folha de eventos, adicione uma variável global do tipo `Number` com o nome de `chave` e inicie-a com `0`. A ideia é que, quando o jogador colidir com a chave no jogo, essa variável fique com o valor `1`. Deste modo, sabemos se ele já a coletou ou não.

Para isso, adicione um evento ao objeto `Jogador` com a condição `On collision with another object`, então escolha a `Sprite chave` no campo `Object`. Vamos criar duas ações para esse evento. Na primeira, escolheremos a `Sprite chave` e a ação `Destroy` para ela, pois, quando o jogador colidir com a chave, queremos que ela desapareça do layout.

Para a segunda ação, vamos escolher o objeto `System` e ação `Set value`. Na janela que abrir, escolha a variável global `chave` e dê o valor `1`, como mostra a figura a seguir. Desejamos que, quando a chave for coletada pelo jogador, a variável global `chave` seja setada para `1`.

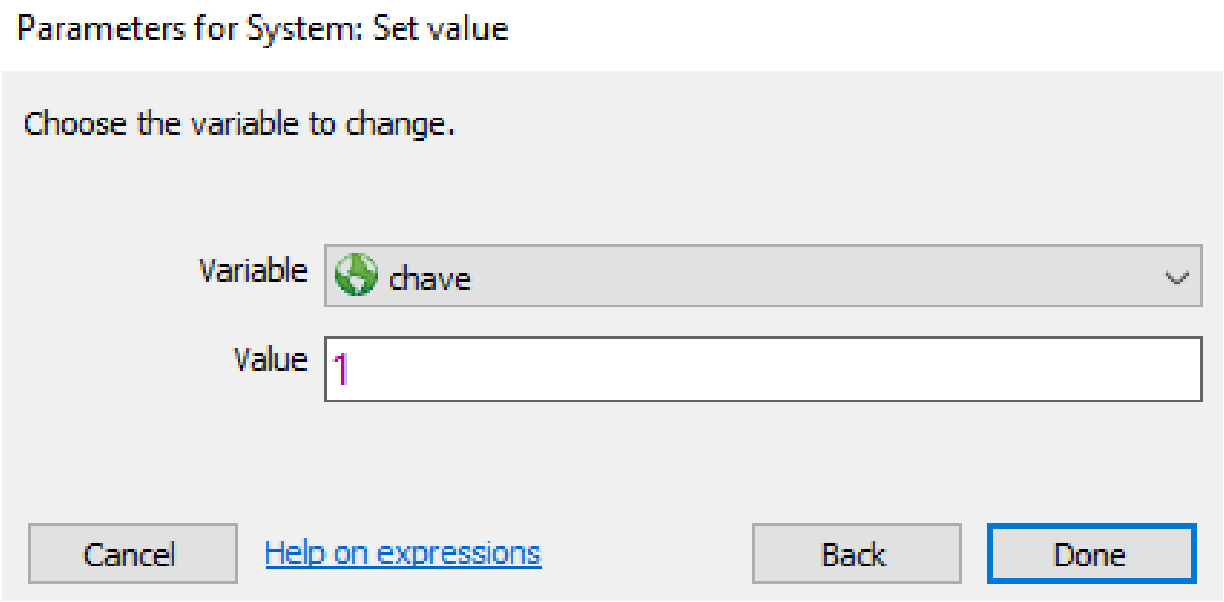


Figura 7.7: Set value

No final, teremos o evento configurado da seguinte maneira:

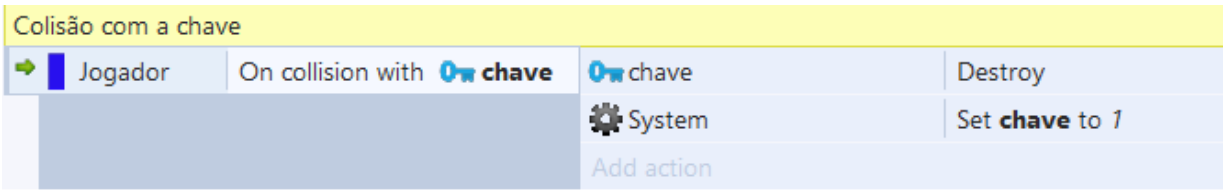


Figura 7.8: Evento de colisão com a chave

Agora que conseguimos saber se o jogador possui a chave ou não pela variável global `chave`, podemos configurar a porta, fazendo-a funcionar ou não. Quando o jogador colidir com a porta e a variável global `chave` for igual a `1` (ou seja, já coletou a chave), queremos que a porta mude sua animação de fechada para aberta.

Para isso, adicione um evento ao objeto `Jogador` com a condição `On collision with another object`, e escolha a `Sprite porta_fase1` no campo `Object`. Isso vai identificar o contato do jogador com a porta. Agora, adicione mais uma condição a esse evento, escolha o objeto `System` e a condição `Compare variable` para ele. Coloque os seguintes parâmetros:

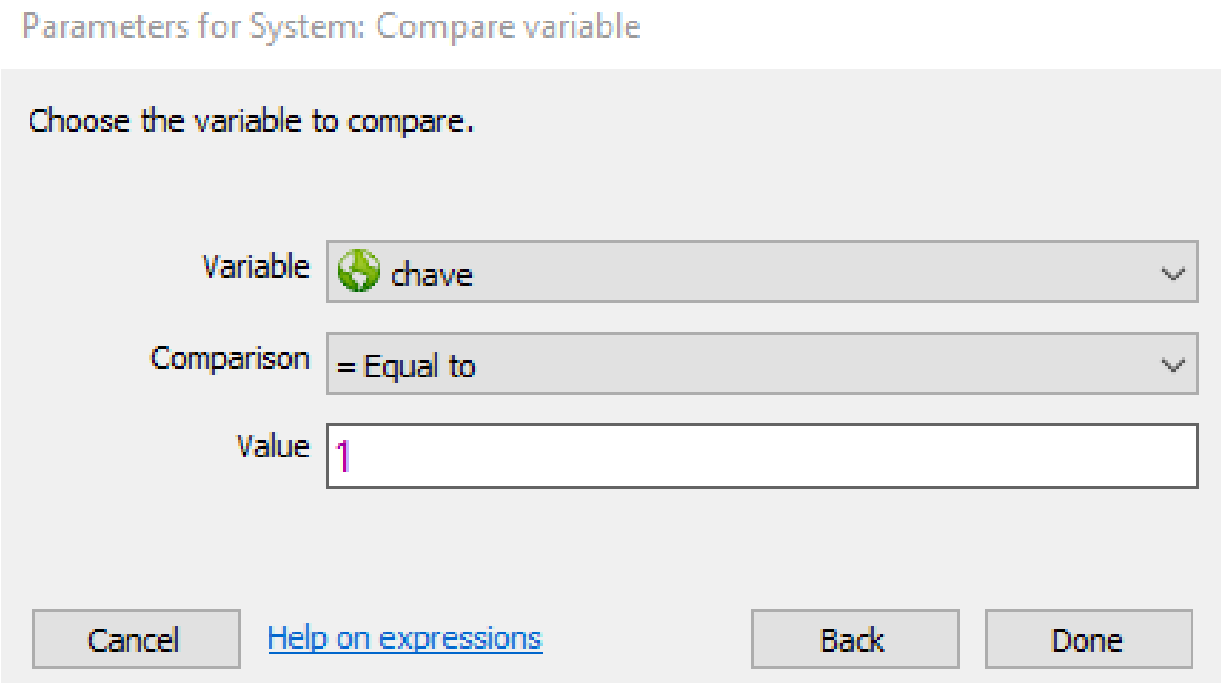


Figura 7.9: Compare variable

Com isso, verificamos se o jogador pegou a chave ao colidir com a porta. Por fim, falta somente trocar o quadro de animação para aquele que representa a porta aberta.

Assim, na ação, escolha a `Sprite porta_fase1` e a ação `Set frame`, para setar a animação. No campo `Frame number`, coloque o valor `1`, que é o número do frame que a representa aberta. O evento de colisão ficará assim:

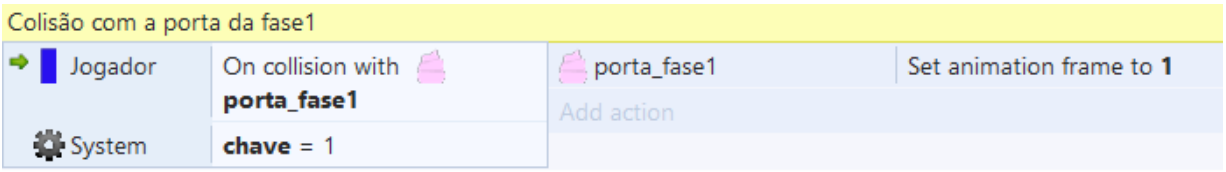


Figura 7.10: Evento de colisão com a porta

Nosso sistema de porta e chave está pronto! Foi bem simples, não?

Nos próximos capítulos, vamos inserir o layout de seleção de fases para que, quando o jogador colidir com a porta, além de ela ser aberta, ele consiga ir para a próxima fase. Mas antes de fazermos isso, vamos continuar com os itens coletáveis e adicionar o sistema de pontuação no jogo.

7.2 Adicionando a pontuação no jogo

Vamos agora programar a pontuação do jogo, que vai ser somada a partir dos itens coletados e inimigos destruídos no decorrer da fase. Primeiramente, importaremos um novo tipo de objeto para o layout, o de tipo `Text`. Este tipo nos permite desenhar textos na tela de forma estática ou dinâmica.

Isso significa que podemos usar esse objeto para exibir a pontuação atual do jogador. Assim, a cada novo ponto que ele fizer, o valor na tela será atualizado.

Para isso, importe o objeto `Text` no layout da layer `UI`, como mostra a figura seguinte. Dê o nome `pontuacao` neste objeto de texto.

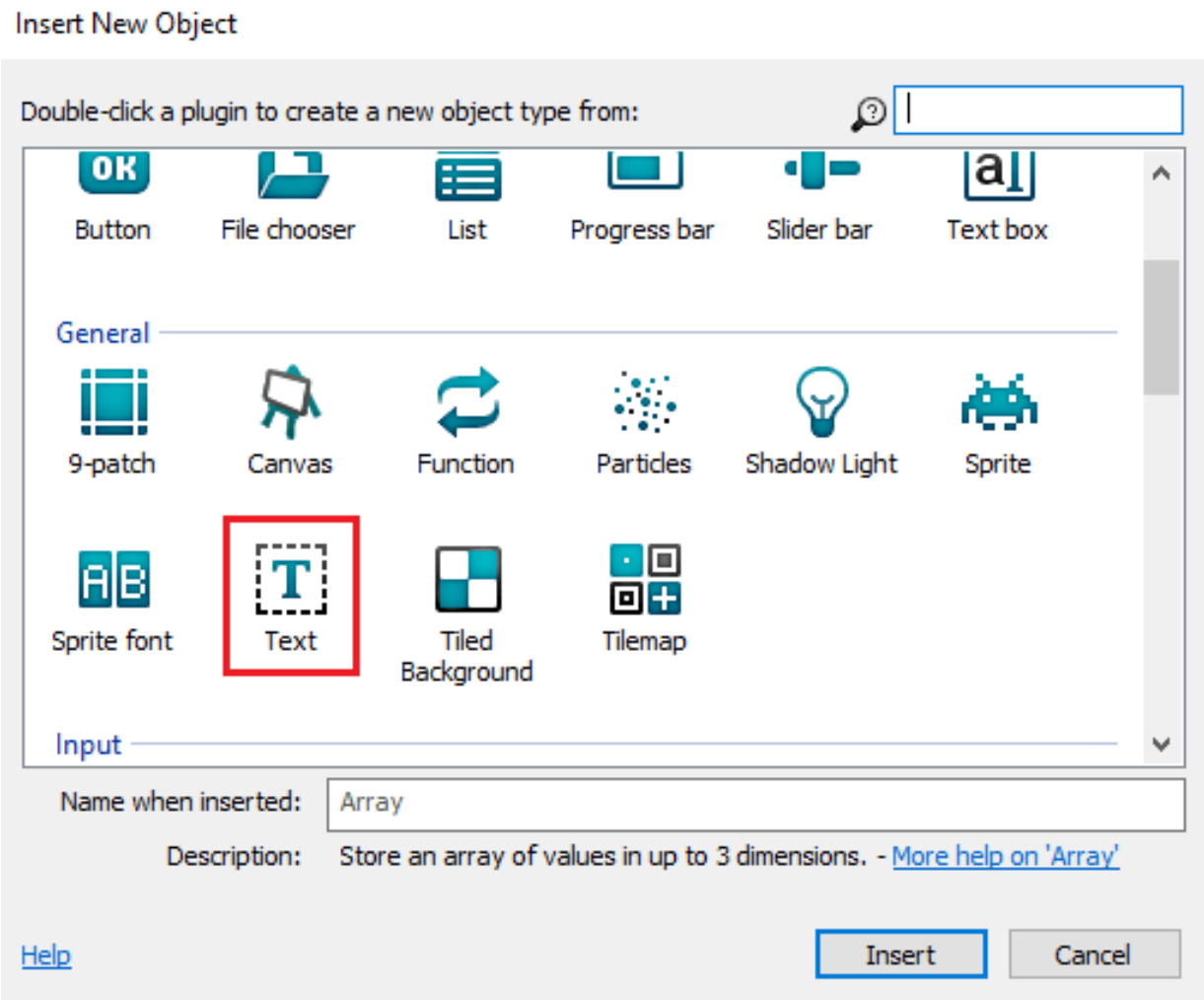


Figura 7.11: Importando o objeto de texto

Nas propriedades desse objeto, colocaremos as informações a seguir para ele se posicionar no canto superior esquerdo do layout.

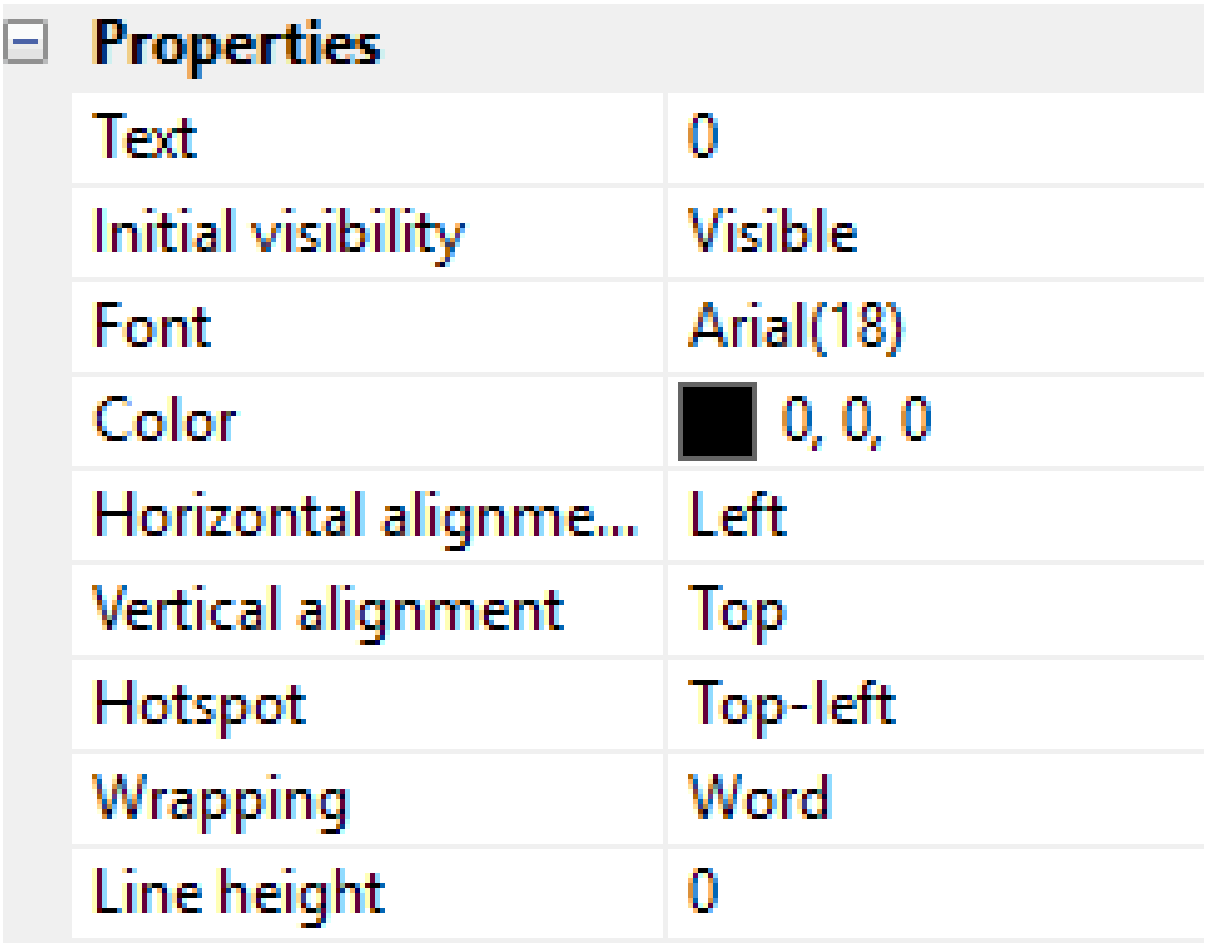


Figura 7.12: Propriedades do objeto de texto

Na barra de propriedades do objeto, adicionaremos uma variável local, como já aprendemos no capítulo *Sistema de vida e adi ão de inimigos*, para o objeto `pontuacao` . Ela será responsável por marcar a pontuação do jogo e transferi-la para o objeto de texto. Para ela, dê o nome de `pont` e valor inicial `0` (zero):

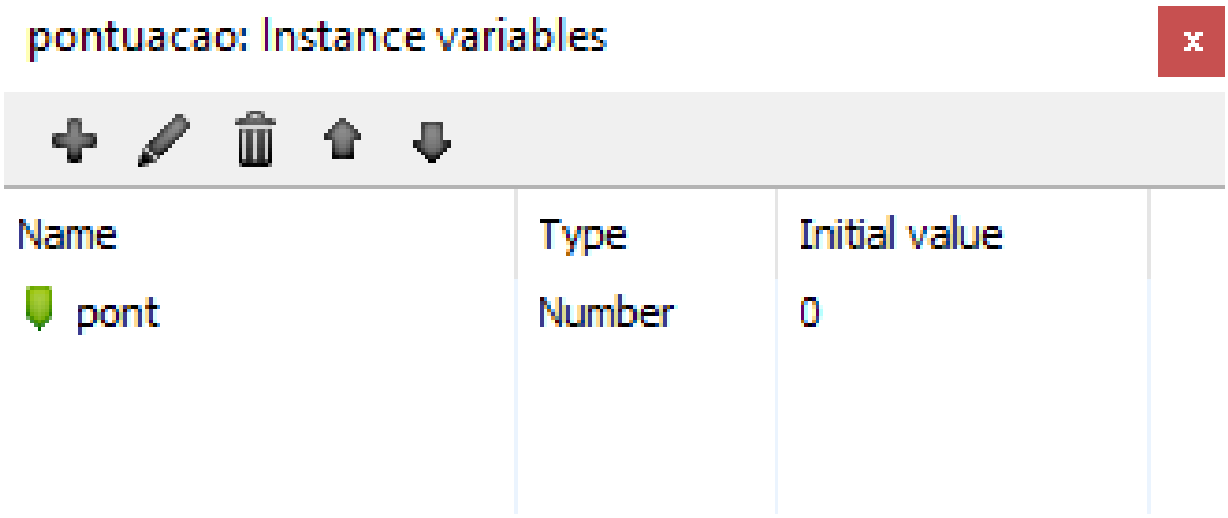


Figura 7.13: Adicionando uma variável local ao objeto de texto

Pronto, agora já temos o nosso contador de pontos funcionando. O próximo passo é fazer com que as ações no jogo gerem pontos. Como já havíamos combinado, as ações serão:

- Coletar moedas;
- Coletar estrelas;
- Eliminar um inimigo.

Vamos começar fazendo a colisão com as moedas; cada uma delas gerará 100 pontos. Para isso, na ação do evento do jogador colidindo com a moeda, escolha o texto `pontuacao` e dê a ação `Add to` para ele. Coloque o valor `100` para a variável local `pont`, como vemos a seguir, pois queremos que sejam somados 100 pontos no momento dessa colisão.

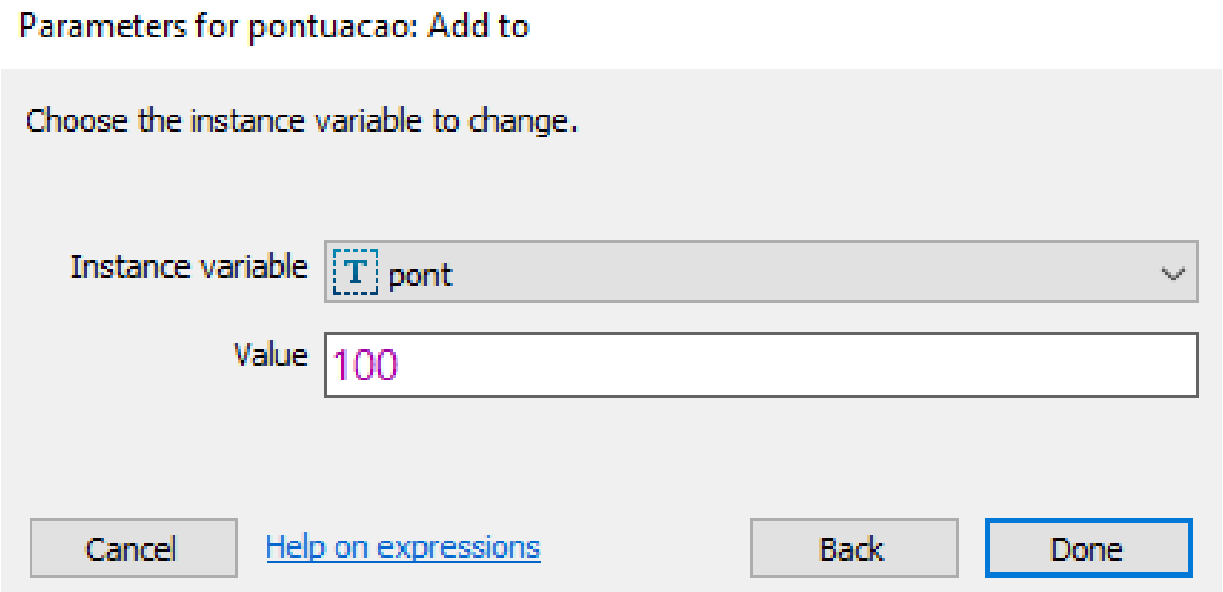


Figura 7.14: Add to

Na segunda ação, vamos "enviar" o valor da variável local `pont` ao texto `pontuacao` , para o valor dos pontos aparecer na tela do jogo. Escolheremos novamente o objeto de texto `pontuacao` e a ação `Set text` . Coloque os parâmetros a seguir para a transferência dos pontos entre a variável e o texto:

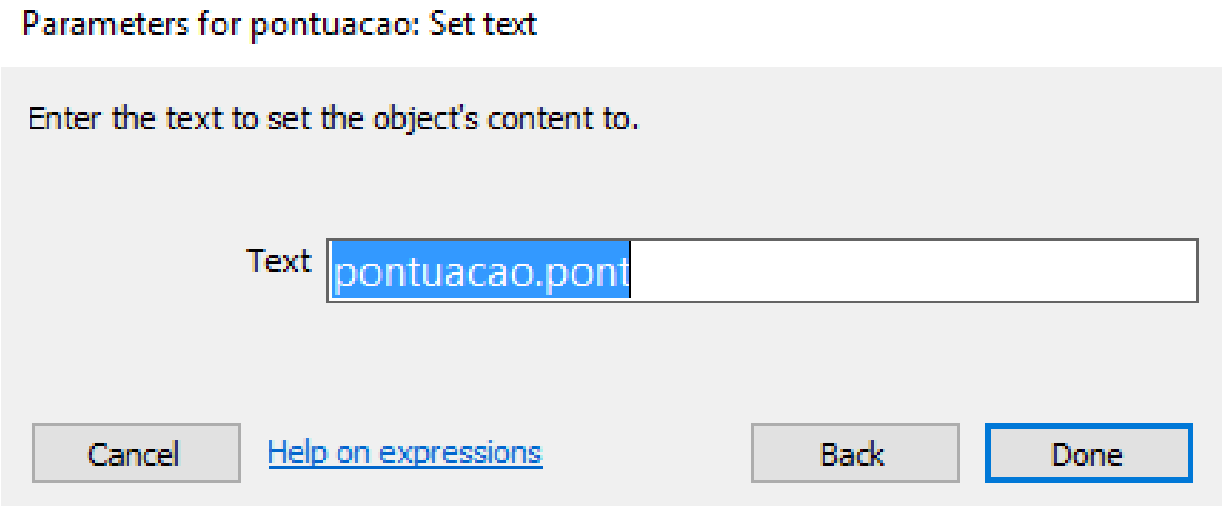


Figura 7.15: Set text

O evento de colisão com a moeda ficará desta forma:

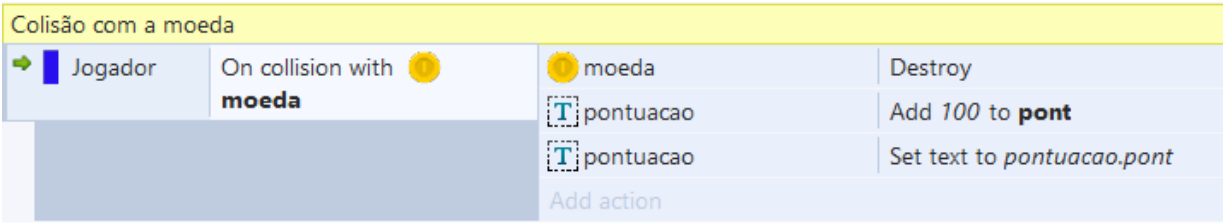


Figura 7.16: Evento de colisão com a moeda

Vamos colocar essas duas linhas de ação inseridas também para o evento de colisão com a estrela e com o `inimigo1`. A figura a seguir mostra como ficarão esses dois eventos e suas ações:

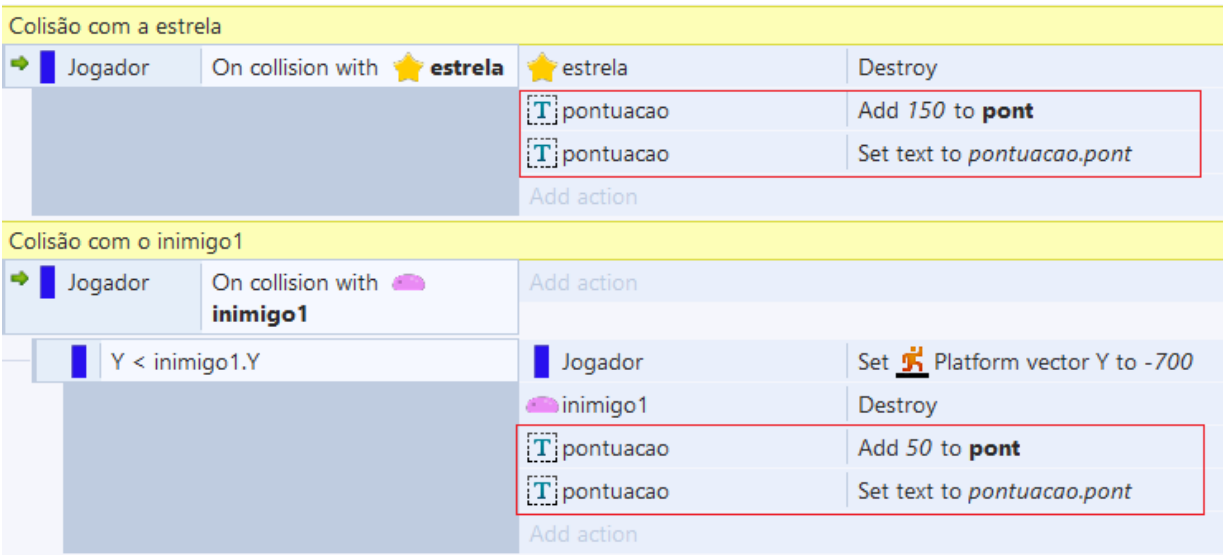


Figura 7.17: Inserindo o evento pontuacao

Note que as linhas são iguais, exceto pela diferença no valor dos pontos. Colocamos 150 pontos para a colisão com uma estrela e 50 pontos para a destruição do inimigo. Estes foram os valores que escolhemos, mas você tem a liberdade de adicionar qualquer outro para essas situações.

Antes da próxima parte do desenvolvimento do jogo, precisamos adicionar mais três ações importantíssimas. Como os valores da pontuação, das vidas e do mecanismo de chave são acumulativos – ou seja, são contabilizados uma vez que o jogador os ganhou –, precisamos resetá-los sempre que o jogador estiver jogando

novamente o layout. Caso contrário, mesmo que ele perca o jogo, a pontuação e a vida serão mantidas.

Para isso, vamos resetá-las no evento `On start of layout`, que criamos anteriormente no capítulo *Movimenta ão e anima ão do personagem*. Estão lembrados? A intenção é que as variáveis globais `chave` e `pontos` sejam setadas com o valor `0` (zero), e a variável local `vida` do `JogadorAnimacao` seja setada com `3` toda vez que o layout for inicializado. Isso porque não queremos que o jogo comece com as variáveis do anterior guardadas nele, nem que o jogador comece com menos de três vidas.

Assim, para a ação do evento de inicialização do layout (`On start of layout`), escolha o objeto `System` e a ação `Set value` para ele. Então, selecione a variável global `pontos` e o valor `0` (zero), conforme a seguir:

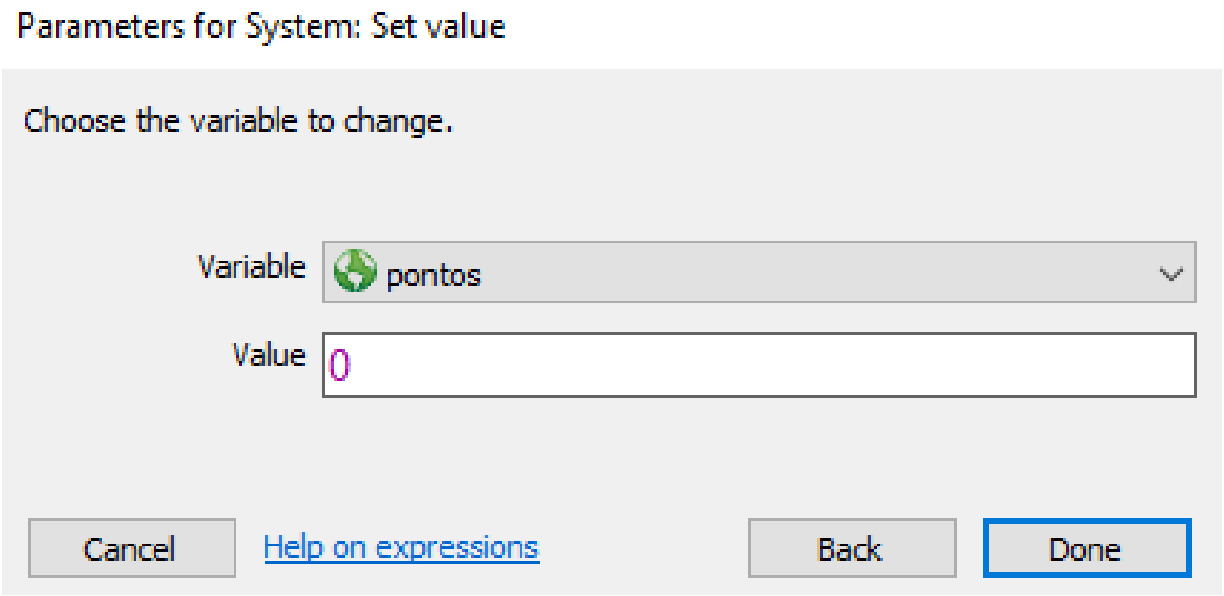


Figura 7.18: Set value na variável global pontos

Faça a mesma coisa com a variável global `chave`. Já para a variável local `vida` do objeto `JogadorAnimacao`, temos de inicializá-la com `3`, pois precisamos que o jogador comece com 3 vidas no início da

fase. Logo, selecione o `JogadorAnimacao` e dê a ação `Set value`. Coloque os seguintes parâmetros nessa aba:

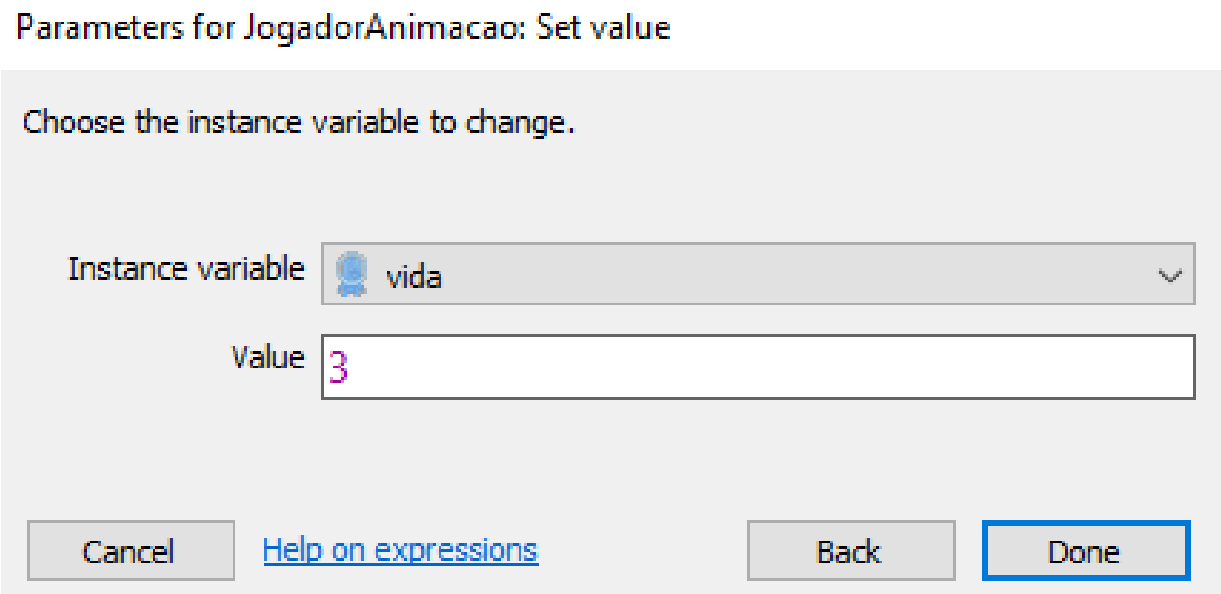


Figura 7.19: Set value na variável local vida

O evento `On start of layout` ficará assim:

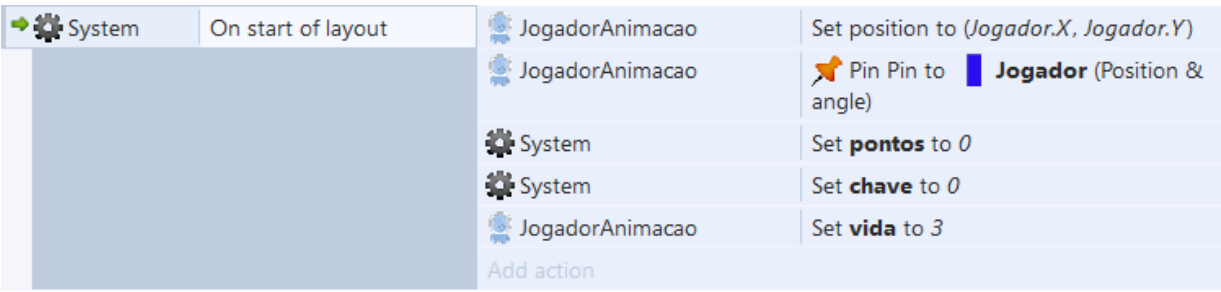


Figura 7.20: Novo evento de inicialização do layout

Pronto, nosso sistema de pontuação está finalmente concluído. Já podemos fazer nosso jogador andar pelo cenário e colidir com os itens para somar pontos. Vamos agora configurar nosso sistema de corações.

7.3 Adicionando os corações no jogo

Agora vamos adicionar os corações ao layout. Eles vão representar as três vidas do jogador. Programaremos para que um coração fique transparente cada vez que o jogador perder uma vida.

Primeiro, vamos adicionar uma `Sprite` na layer `UI` com o nome de `coracoes`. Importe as quatro imagens `Corações` da nossa pasta de recursos nos quadros de animação, e coloque-as na ordem correta. Isto é, no quadro zero, coloque a imagem de zero coração; no quadro 1, coloque a imagem de 1 coração, e assim sucessivamente até o terceiro quadro, como a seguir:

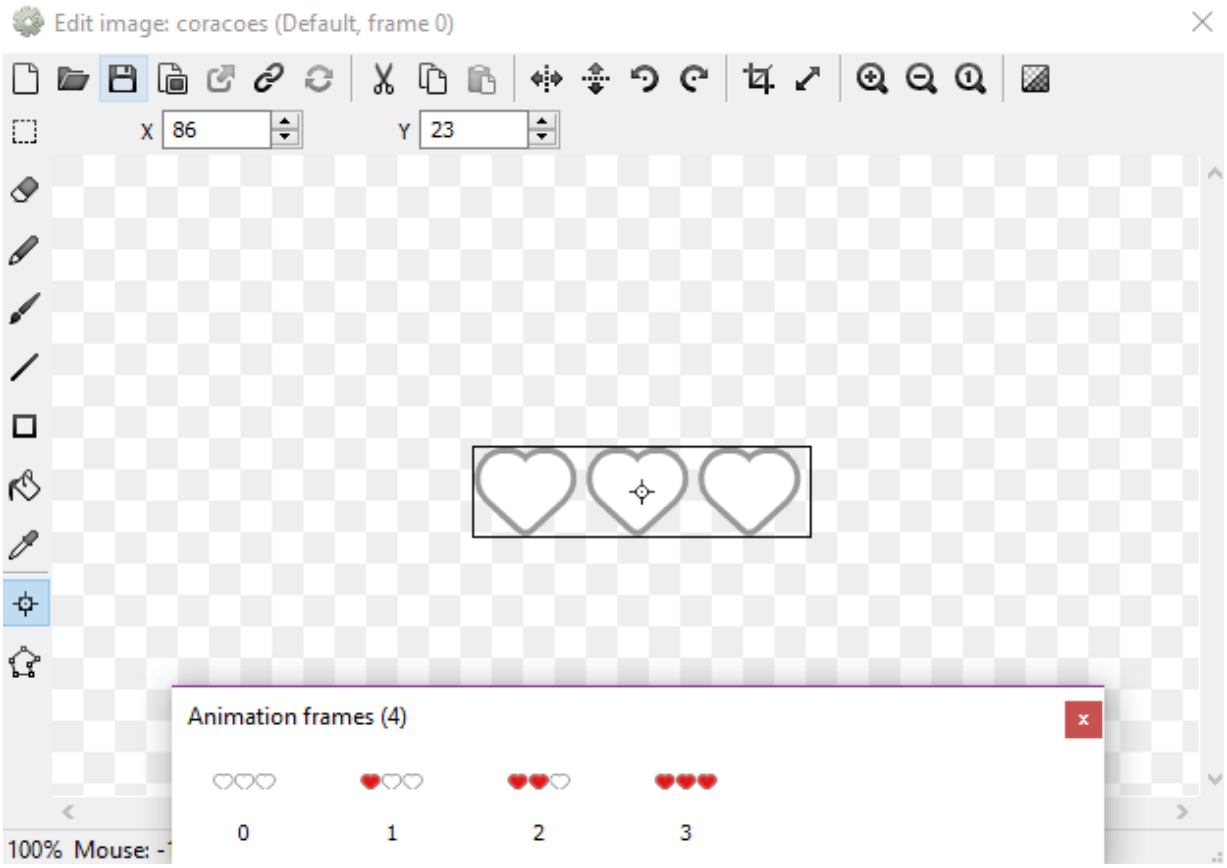


Figura 7.21: Sequência dos frames dos corações

Essa ordem é necessária para podermos programar os eventos corretamente, como veremos a seguir.

Não se esqueça de mudar a velocidade (`speed`) na barra de animação para zero, pois não queremos criar nenhuma animação com esses frames. Posicione a `Sprite coracoes` no canto esquerdo superior da fase, abaixo do texto `pontuacao` :

Figura 7.22: Corações inseridos no layout

Vamos complementar alguns eventos responsáveis por subtrair um ponto na vida do jogador, que fizemos anteriormente. Esses eventos são os de colisão com o `inimigo1` , o `inimigo2` e o `inimigo3` . Queremos que uma vida seja reduzida e o frame da `Sprite coracoes` seja trocado, para indicar que uma vida foi perdida, cada vez que o jogador colidir com um dos inimigos.

Para isso, vamos escolher para a ação desses eventos o objeto `Sprite coracoes` que acabamos de criar e dar a ação de `Set Frame` .

No campo `Frame number`, precisamos acessar a variável local `vida` do `jogadorAnimacao`. Para conseguirmos acessar a variável local de um objeto, precisamos colocar o nome do objeto, seguido de um ponto, e o nome da variável. No nosso caso, `jogadorAnimacao.vida`, como mostra a figura a seguir:

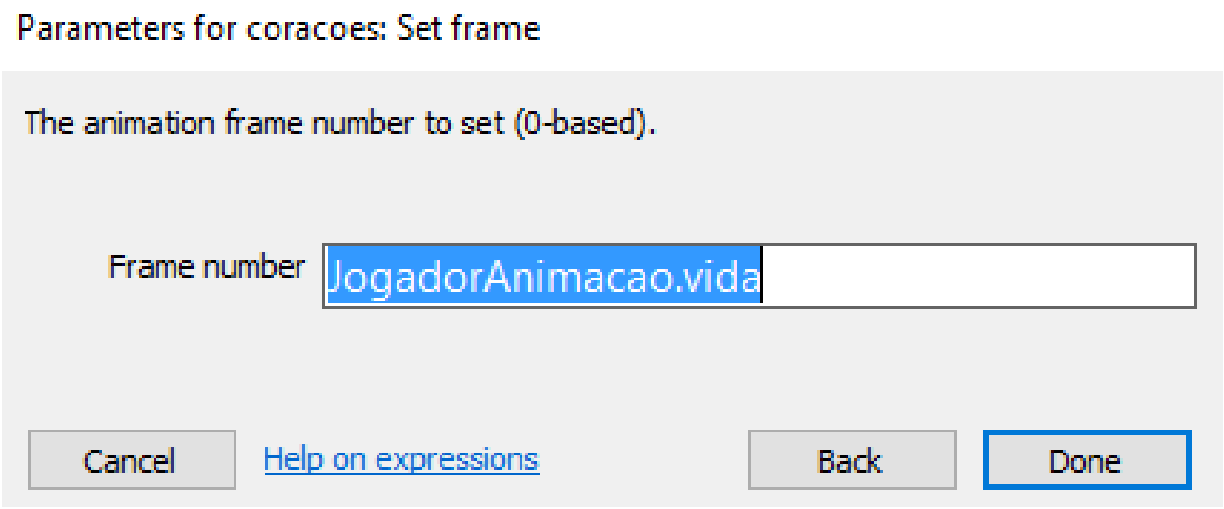


Figura 7.23: Set frame para o objeto dos corações

Isso fará com que o frame da `Sprite coracoes` mude de acordo com o valor que a variável local `vida` está guardando. Ou seja, se a variável `vida` estiver com o valor `2`, o frame da `Sprite coracoes` será trocado automaticamente para o 2.

Vamos colocar essa mesma linha de código nos três eventos referentes aos inimigos. A figura seguinte mostra como ficará a folha de eventos.

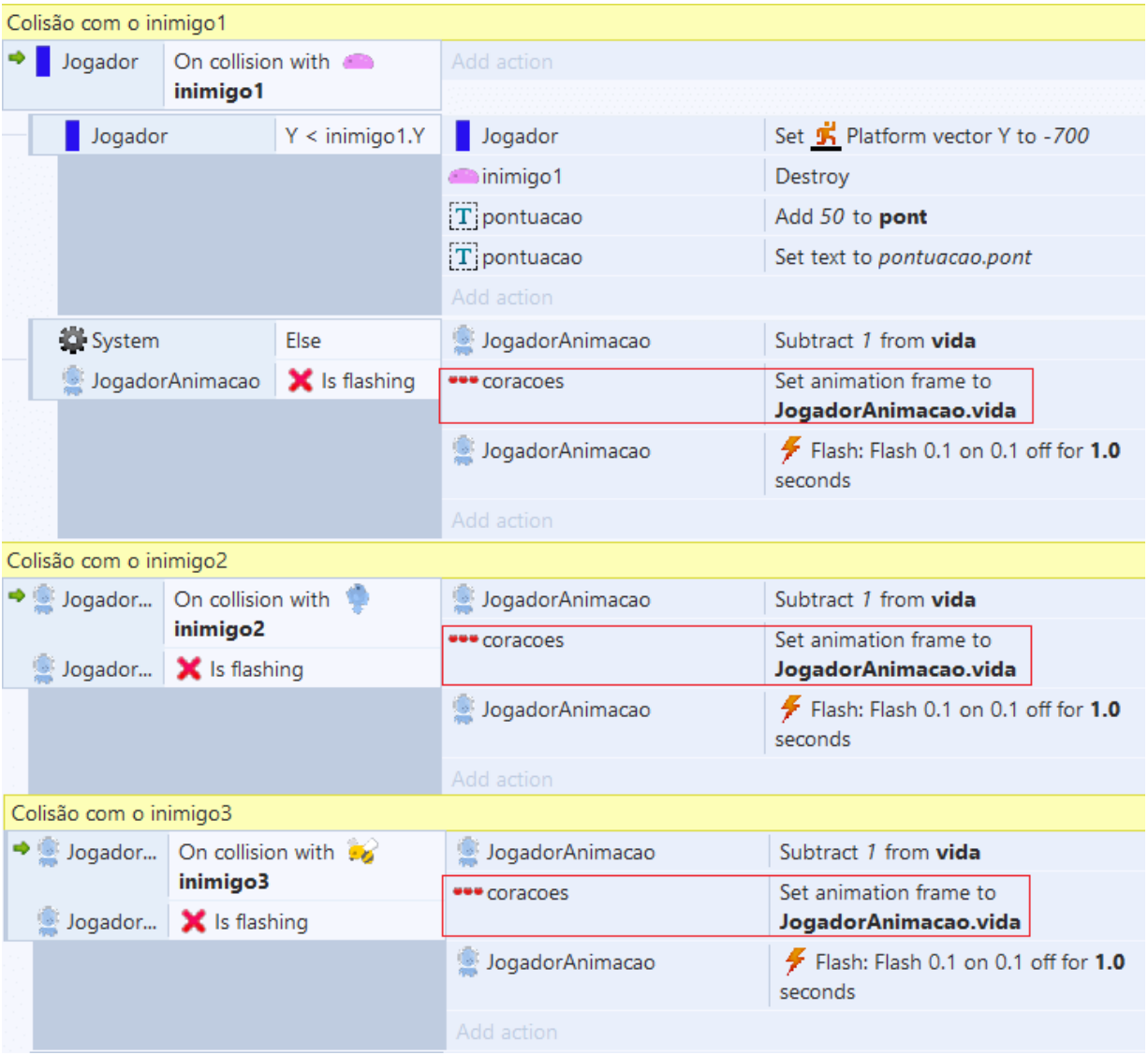


Figura 7.24: Eventos inimigos

Pronto! Finalizamos toda a parte da pontuação do nosso jogo. Adicione moedas, estrelas e inimigos no layout e veja o mecanismo programado funcionar, tanto no contato com os itens coletáveis quanto na pontuação atualizada na tela.

No próximo capítulo, vamos inserir alguns efeitos para nosso jogo ficar ainda mais bacana e criar novos layouts.

Capítulo 8
Criando efeitos e outros Layouts

Chegamos à parte de deixar o nosso jogo mais interessante! Neste capítulo, vamos inserir um efeito de chuva no nosso layout, somente para diferenciar nossas fases e deixá-las mais bonitas. Esse efeito será criado a partir do sistema de efeitos e partículas do Construct 2.

Também criaremos um efeito de água para representar os rios que servirão de obstáculos nas fases. A ideia é que, quando o jogador cair dentro da água, ele perca uma vida e volte para o início da fase. Logo em seguida, começaremos a trabalhar em novos layouts para o jogo. Prontos?

8.1 Inserindo partículas

O objeto `Particles` (partículas) permite gerar efeitos visuais, criando e movendo muitas imagens pequenas de forma independente dentro do cenário. Assim, é possível criar diversos tipos de efeitos, como chuva e neve, por exemplo. Muitas vezes, um simples ponto colorido em um fundo preto é suficiente para criar um efeito bem bacana.

Confira a figura a seguir, em que temos um objeto de partícula de um simples ponto colorido ao lado esquerdo, e é mostrada essa mesma partícula no navegador ao lado direito.

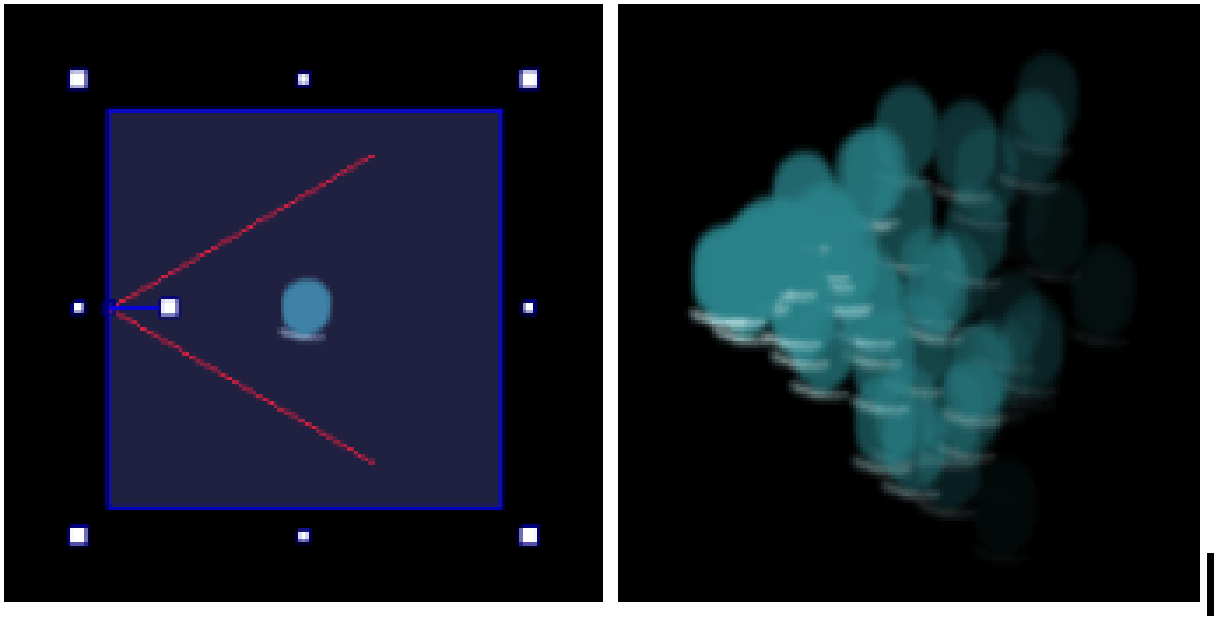


Figura 8.1: Adicionando partículas

A criação de partículas costuma ser um trabalho bastante complicado. Em plataformas como o Unity ou o Unreal, este processo pode ser bastante complexo. Felizmente, o Construct 2 faz todo este trabalho pesado para nós. Tudo isso é feito por meio do sistema de partículas.

No nosso jogo, vamos inserir um efeito de chuva. Para isso, usaremos o objeto `Particles` . Na aba de inserção de objetos, selecione-o, como mostra a figura:

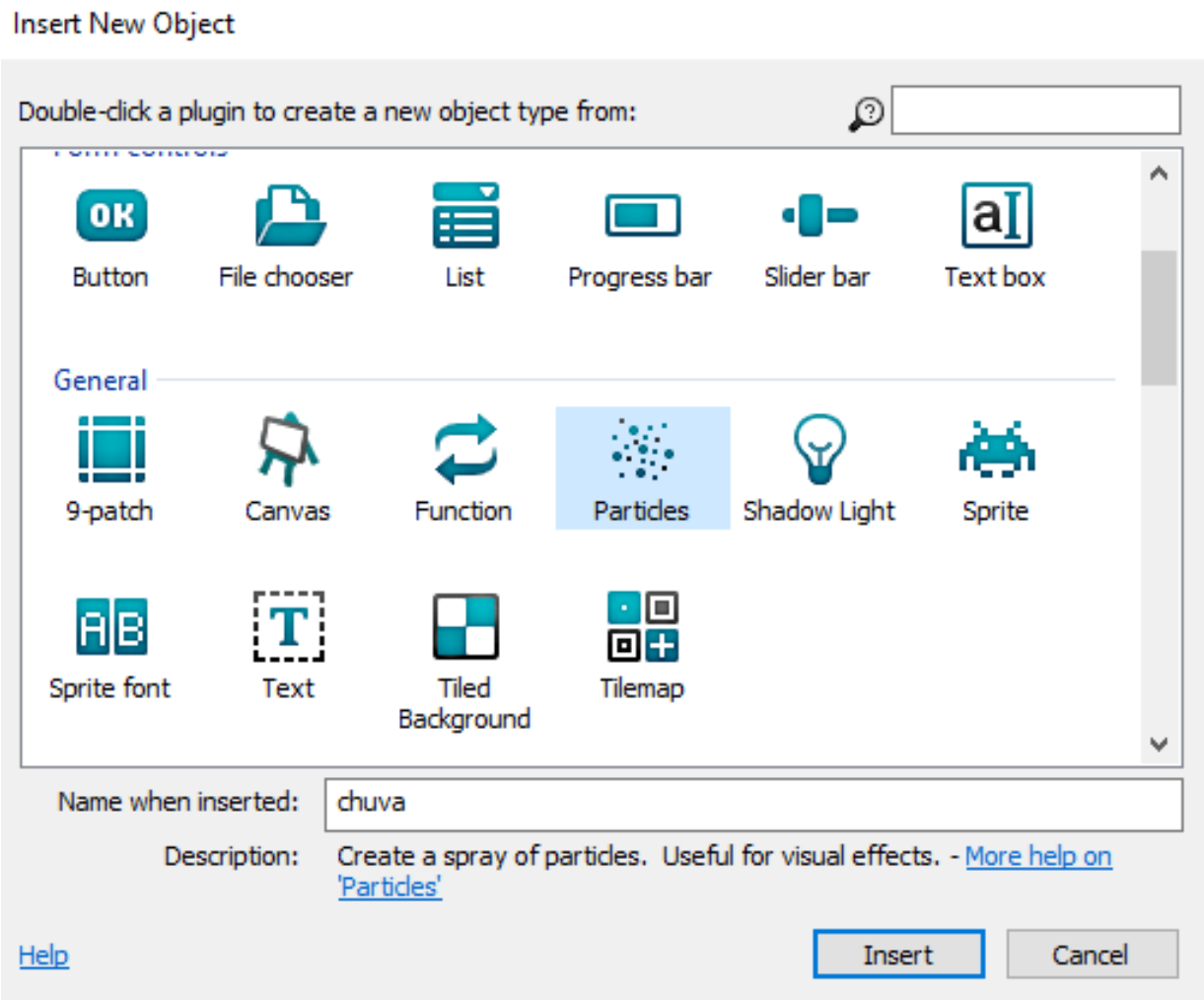


Figura 8.2: Inserindo um objeto de partícula

Insira-o na camada de nome `Principal` e nomeie-o de `chuvaParticula`. No editor de imagens, importe a imagem `pingo` do diretório `Partícula` da nossa pasta de recursos. Use a ferramenta `resize` do editor para ajustá-la para 14x22 pixels, pois este é um bom tamanho para um pingo de chuva em relação ao tamanho do nosso layout. Agora coloque-o acima do layout, como mostra a figura:

Figura 8.3: Objeto de partícula inserido acima do layout

Nas propriedades da partícula, vamos primeiramente mudar o ângulo (`Angle`) para `90` graus. Isso fará com que os pingos caiam no sentido de cima para baixo. Para configurar a quantidade de pingos que cairão na tela, vamos mexer na propriedade `Rate` . Ela é responsável por definir o número de partículas criadas por segundo. Para criar um efeito de chuva bacana, sem poluir a tela, insira o valor `500` .

Agora precisamos configurar o ângulo de curvatura de onde sairão os pingos. Repare que, por padrão, este valor é de 60 graus. Esse

ângulo representa o alcance em graus das partículas. Como queremos que as partículas caiam em todo o cenário, precisamos configurar essa abertura para 180 graus. Veja só a diferença na figura:

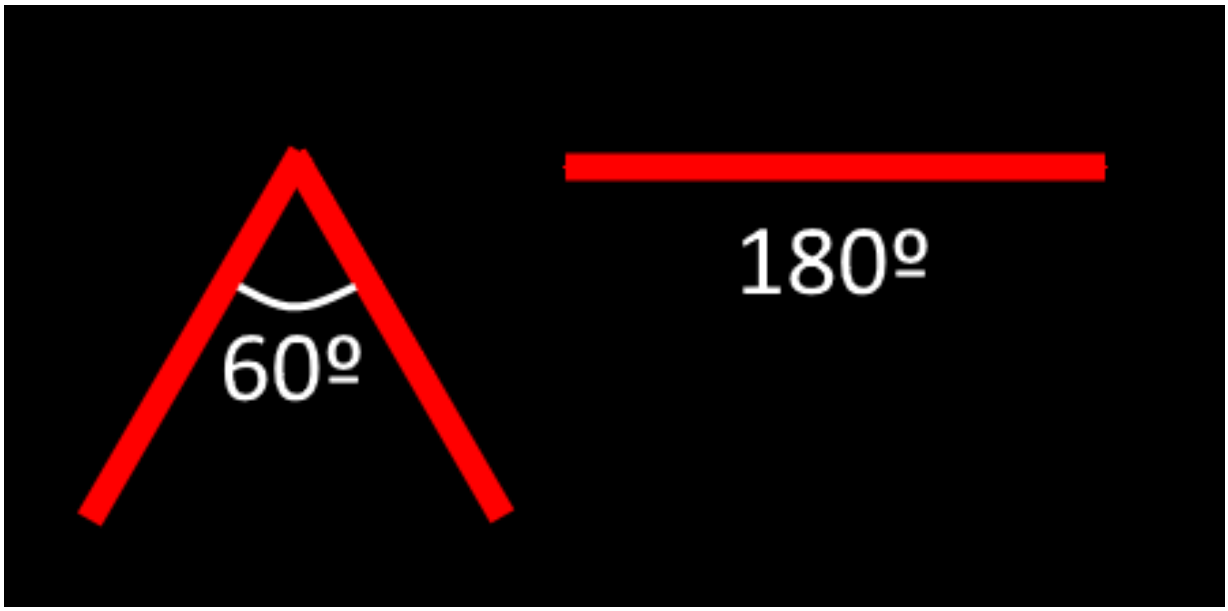


Figura 8.4: Ângulos de curvatura

Feito isso, precisamos configurar a velocidade das partículas. Isso é feito na propriedade `Speed`. Essa velocidade influencia na intensidade da geração das partículas no cenário. Como vamos criar um efeito de chuva fraca, colocaremos a velocidade em `10` por segundo. Para o tamanho inicial das partículas, colocaremos `10` na propriedade `Size` também.

Para fazer a chuva cair em todo o layout, vamos utilizar a propriedade `X randomizer`. Ela define o deslocamento das partículas no eixo X do plano cartesiano. Como choverá por todo o caminho, colocaremos o tamanho do layout nessa propriedade. Por isso, coloque `6000`.

Para tornar o efeito de chuva ainda mais real, vamos adicionar um fator randômico na velocidade das partículas, para que elas caiam em tempos diferentes. Para isso, coloque o valor `150` na

propriedade `Speed randomiser`. Esse valor deixa uma sensação de velocidade bacana, sem deixar o efeito estranho. Se colocarmos um valor muito alto, algumas partículas ficarão super-rápidas e outras, superlentas. O ideal é que tenhamos um equilíbrio.

Por fim, vamos configurar a gravidade e a aceleração aplicadas nas partículas. Elas afetam a intensidade da chuva. Como queremos criar um efeito de chuva fraca, colocaremos alguns valores que julgamos bacanas. Fique à vontade para alterar e tornar o efeito a sua cara, ok?

Para configurar a aceleração da chuva, coloque o valor `1` na propriedade `Acceleration`. Assim como na física, a aceleração determina a taxa de variação da velocidade em função do tempo. Em outras palavras, ela indica o aumento ou a diminuição da velocidade com o passar do tempo.

Para a gravidade, coloque o valor `5000` na propriedade `Gravity`. A gravidade é a força que "puxa" as gotas para baixo. Com essa combinação de gravidade, aceleração e velocidade, alcançamos um efeito bonito. Para finalizar e tornar o efeito ainda mais legítimo, tornaremos a opacidade de cada partícula diferente uma da outra.

Conseguimos esta proeza com a propriedade `Opacity randomiser`. Ela funciona de forma semelhante ao `Speed randomiser`. Ela acrescenta um valor randômico à opacidade da partícula, deixando-a mais clara ou transparente. Insira o valor `10` para ela alterar levemente a transparência de algumas das partículas.

Queremos que os pingos sejam destruídos depois de algum tempo, para evitar que o jogo fique lento. Portanto, em `Destroy mode`, escolha a opção `Timeout expired` e, em `Timeout`, coloque o valor `2`. Isso fará com que eles sejam destruídos sempre após 2 segundos de sua criação. Com todos esses valores nos campos das propriedades da partícula, teremos o seguinte:

Layer	Principal
Angle	90
Opacity	100
⊞ Position	373, -448
⊞ Size	128, 128

⊟ Properties	
Rate	500
Spray cone	180
Type	Continuous spray
Image	Edit
⊟ Initial particle properties	
Speed	10
Size	10
Opacity	100
Grow rate	0
X randomiser	6000
Y randomiser	0
Speed randomiser	150
Size randomiser	0
Grow rate randomi...	0
⊟ Particle lifetime properties	
Acceleration	1
Gravity	5000
Angle randomiser	0
Speed randomiser	1
Opacity randomiser	10
Destroy mode	Timeout expired
Timeout	2

Figura 8.5: Propriedades da partícula de chuva

Pronto, já podemos testar o jogo e ver a simulação de chuva funcionando!

8.2 Adicionando efeitos

Agora que vimos um pouco do funcionamento das partículas, vamos partir para os efeitos. Os efeitos (`effects`) têm a função de alterar a aparência visual dos objetos em cena. Eles podem também ser adicionados a camadas ou layouts inteiros.

Entretanto, os efeitos que se misturam com o plano de fundo não podem ser usados em layouts, pois o plano de fundo acabará anulando-o. Na figura a seguir, temos um exemplo de uma `Sprite` com o efeito `Water` (água) ao lado esquerdo e uma `Sprite` sem efeito ao lado direito.

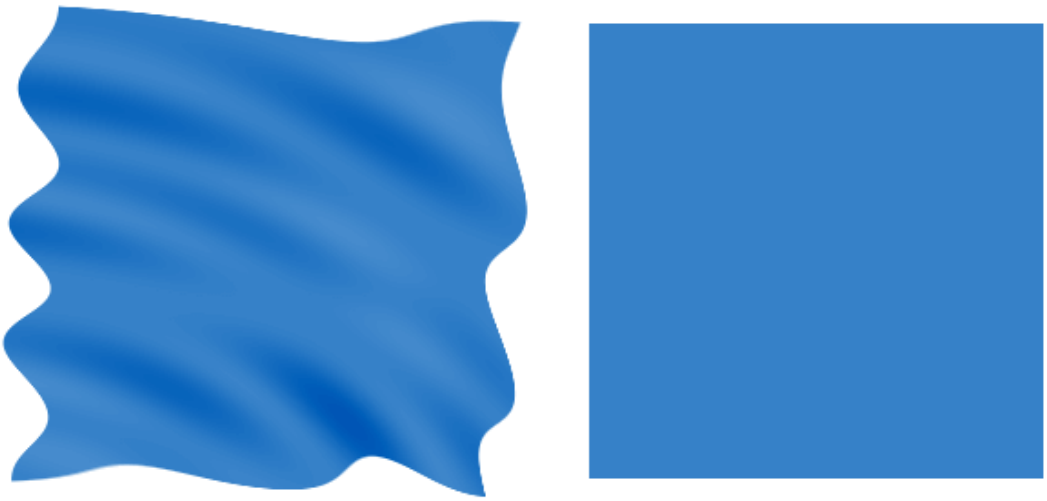


Figura 8.6: Diferença de imagem com e sem efeito de água

Conseguimos adicionar vários efeitos em um único objeto, camada ou layout. Quando isso acontece, os efeitos são acorrentados. Isto é, o segundo efeito só vai começar quando o primeiro terminar, e o terceiro só começará quando terminar o segundo, e assim por diante.

O Construct 2 contém uma ampla biblioteca com mais de setenta efeitos. Um grande número de exemplos de efeitos é mostrado na pasta de exemplos da plataforma, conforme já mencionamos.

Para o nosso jogo, vamos adicionar um efeito de água para representar um "rio" no `Layout Fase1` que estamos criando. Para isso, adicione primeiramente uma `Sprite` com uma cor azul clara e o nome de `agua` na layer `fundo` , conforme a seguir:

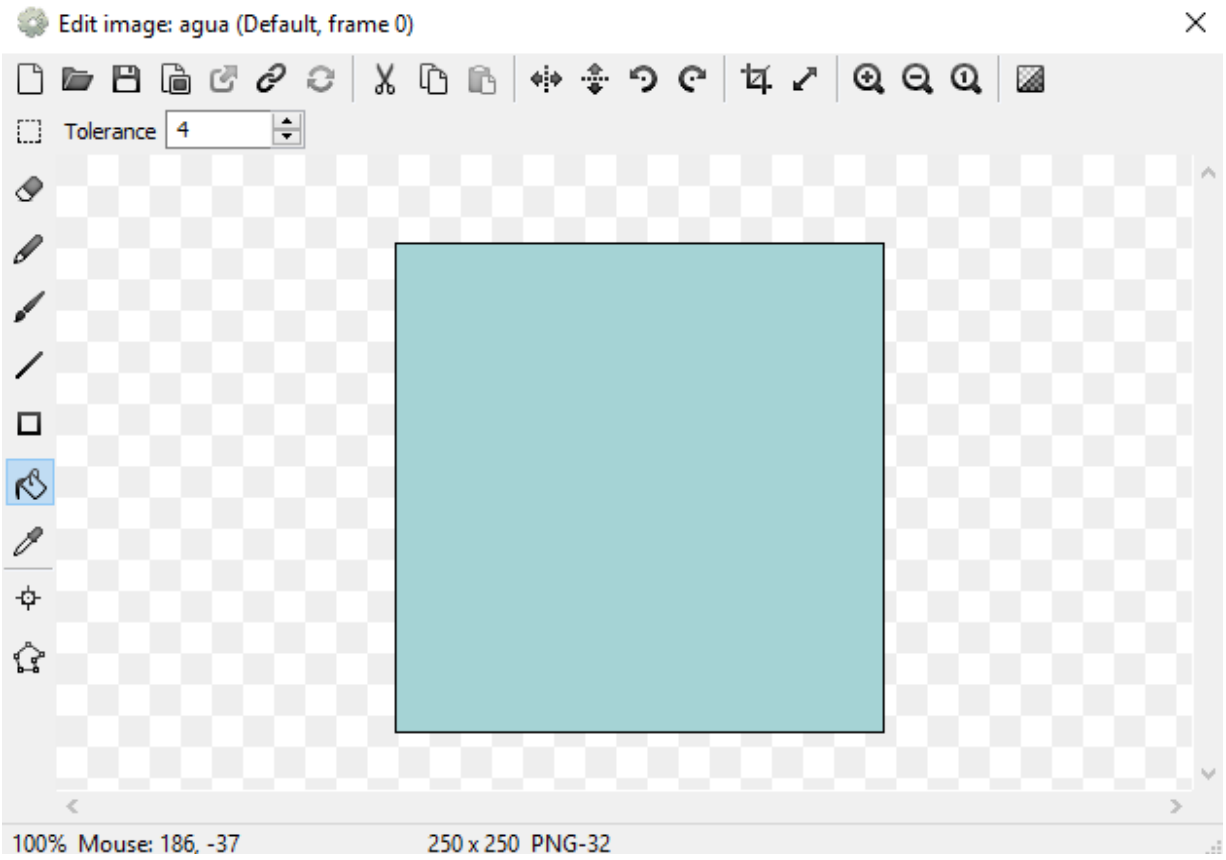


Figura 8.7: Adicionando uma Sprite para representar a água

Coloque-a no layout, clique sobre ela e vá até a barra de propriedades do projeto. Então, escolha a opção de `add/edit Effects` e clique em `+` na aba `Effects`. Em seguida, a caixa de diálogo `Add effect` abrirá, e nela será possível adicionarmos os efeitos. Escolha o `WaterBackground` para dar um efeito de água:

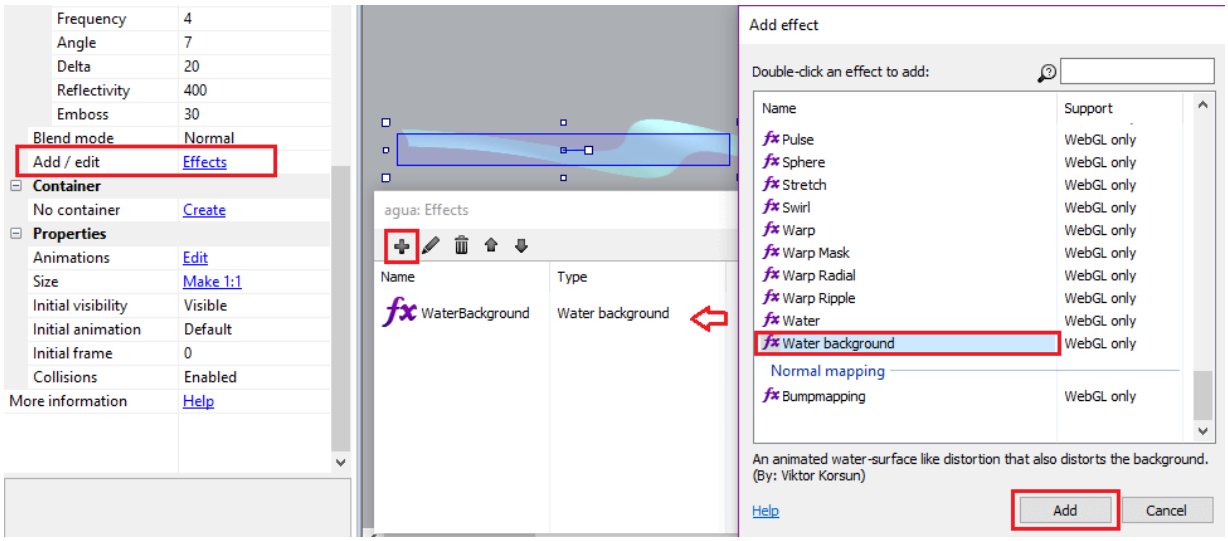


Figura 8.8: Efeito de água

Você vai perceber que a imagem `agua` já foi modificada, porém, não queremos o efeito deste jeito, pois temos uma "onda" grande nesta imagem e desejamos deixá-la mais suave para o nosso rio. Portanto, vá até a barra de propriedades novamente e coloque os seguintes parâmetros no efeito `WaterBackground` , fazendo-o representar ondas suaves:

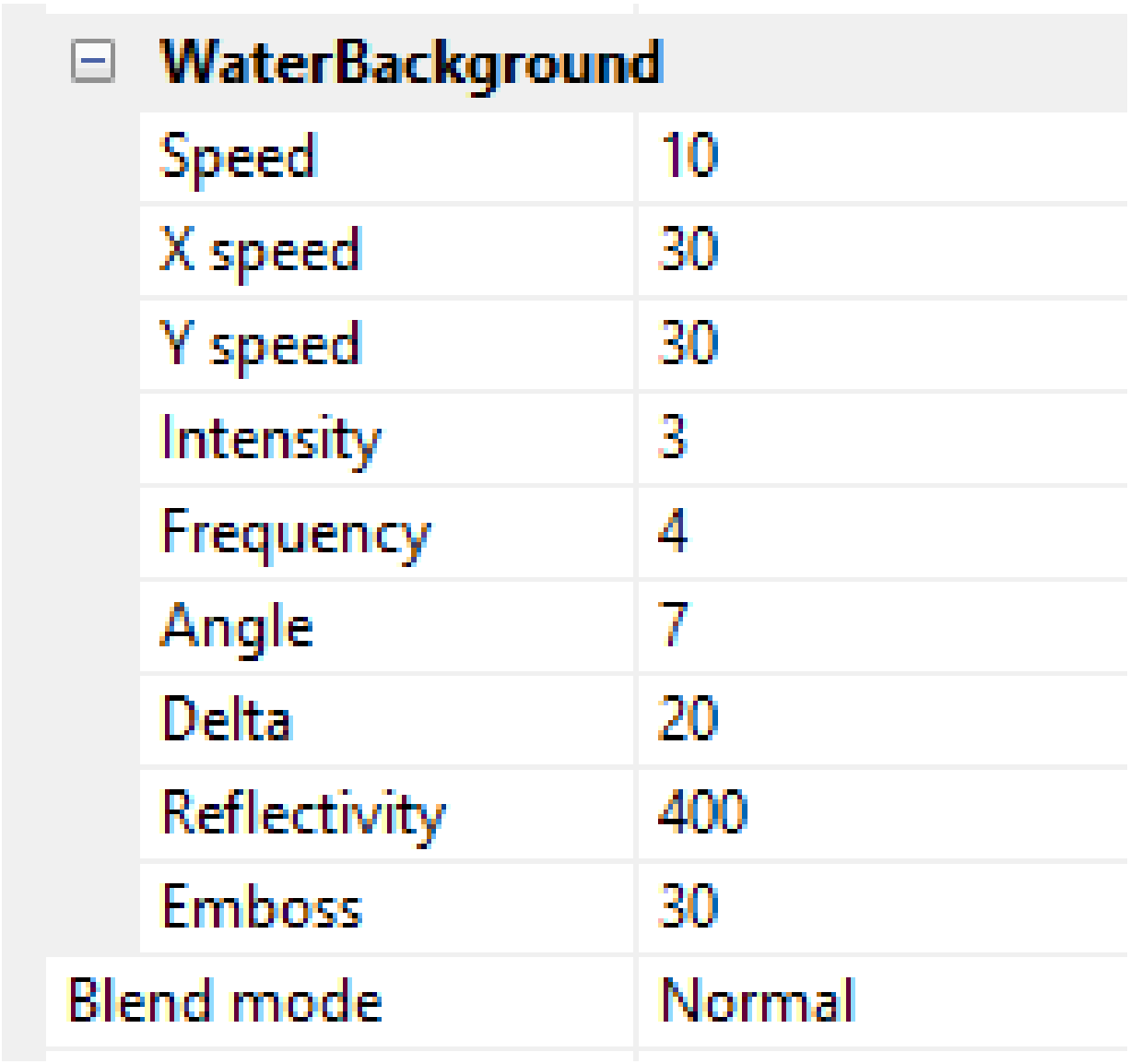


Figura 8.9: Propriedades do efeito WaterBackground

Estes valores foram obtidos por meio da experimentação. Fique à vontade para alterá-los e deixar o resultado que mais gostar. Porém, independente do valor, podemos colocar essa `Sprite` com efeito de água em vários lugares do layout, como vemos na figura:

Figura 8.10: Efeito de água inserido no layout

Assim, criamos o nosso efeito de água!

Neste ponto, já temos o nosso primeiro layout praticamente pronto. Inserimos o jogador, as plataformas, os inimigos, as moedas e as estrelas, e trabalhamos com todos os sistemas necessários para movimentação, colisão e pontuação. Aos poucos, estamos terminando o projeto.

Não se preocupe, pois ainda veremos como funcionam os botões, as músicas e os sons nos próximos capítulos. A seguir, vamos começar a inserir mais layouts de telas e fases no nosso jogo.

8.3 Criando outros layouts

Agora que desenvolvemos bastante o nosso `Layout Fase1`, vamos inserir mais três layouts em nosso jogo: um para a tela inicial, um para a seleção de fases e, por fim, outro para a segunda fase.

Antes disso, vamos colocar mais um evento em nossa `Event Sheet`: o jogador perderá uma vida e voltará ao início do jogo toda vez que ele sair do layout (ou seja, cair dentro do rio). Escolha a `Sprite Jogador` na janela `Add event` e dê a condição `Is outside layout` para ela:

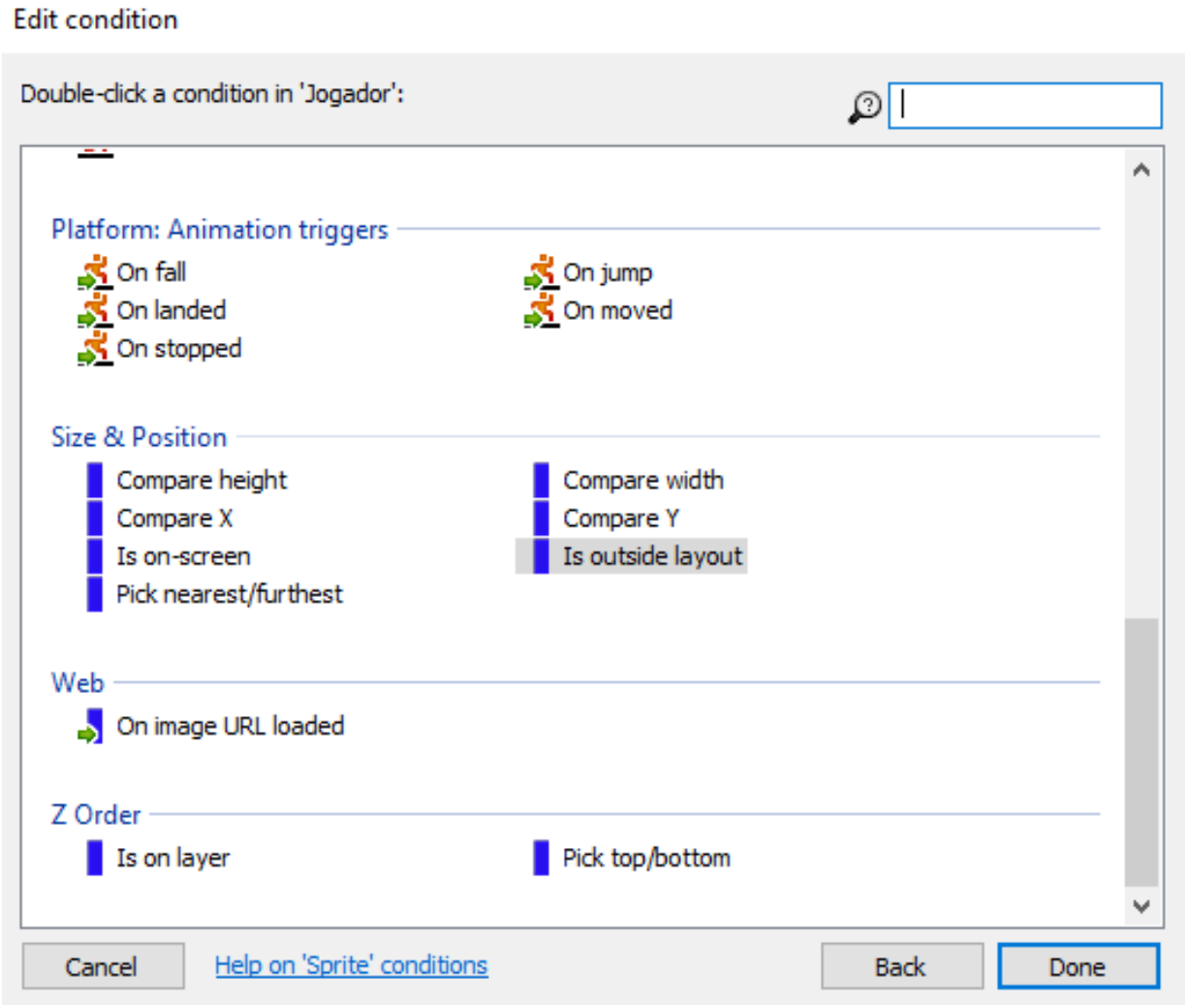


Figura 8.11: Condição Is outside layout

Para essa ação, vamos escolher aquelas duas ações que subtraem o valor `1` da variável local `vida` do `JogadorAnimacao` e setam o frame

`coracoes` com o mesmo valor dessa variável. Para isso, escolha a `Sprite JogadorAnimacao` e dê a ação `Subtract from` com parâmetro `1` para a variável local `vida`. Para a próxima ação, escolha a `Sprite coracoes`, a ação `Set frame` e o parâmetro `JogadorAnimacao.vida` no campo `Frame number`.

A última ação será responsável por fazer o jogador voltar ao início do layout toda vez que sair dele. Para isso, selecione o `Jogador` e dê a condição `Set position` para ele, colocando o valor 94 para X e 454 para Y, como mostra a figura a seguir.

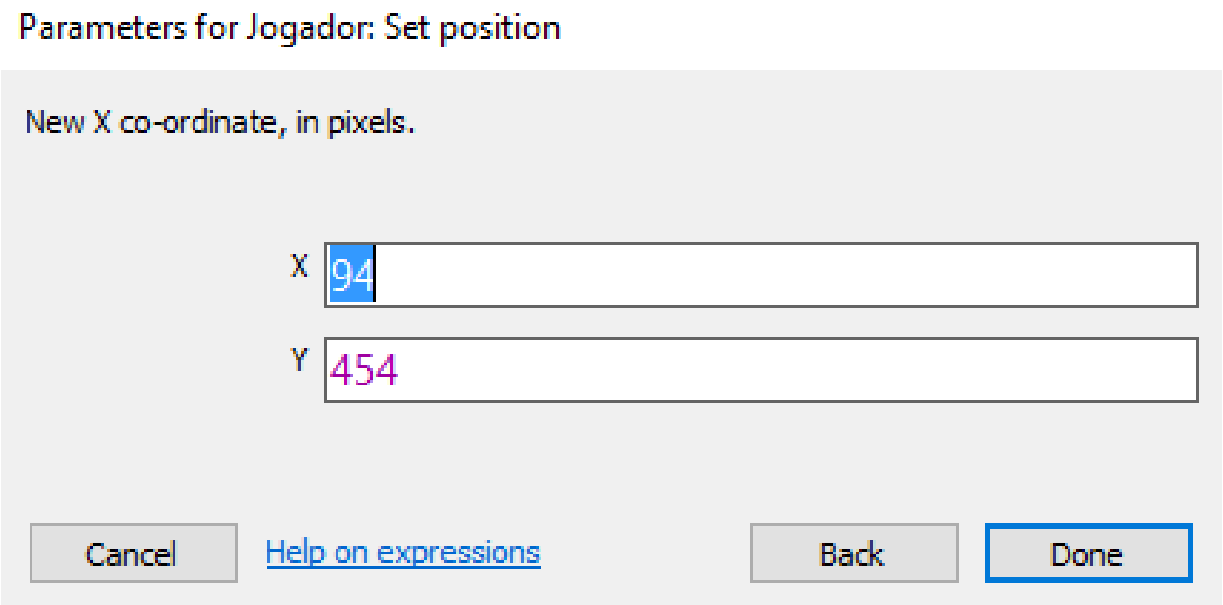


Figura 8.12: Ação Set position

Note que esses valores de X e Y são os mesmos que a `Sprite Jogador` tem no começo do jogo. Se você quiser que ele "renasça" em outro lugar, basta colocar a posição desejada em X e Y.

O seu evento ficará parecido com este:

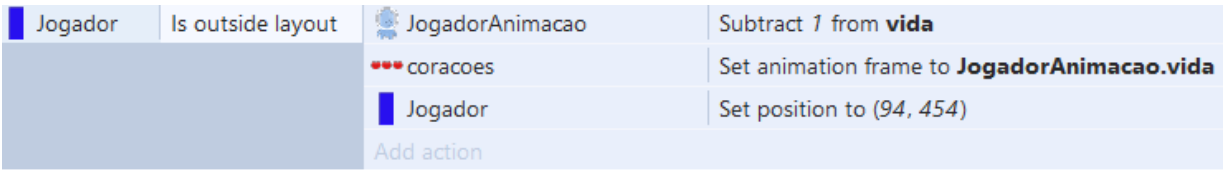


Figura 8.13: Evento do jogador saindo do layout

Tela inicial

Vamos criar agora mais um layout para ser a tela inicial do nosso jogo. Para isso, clique com o botão direito do mouse na pasta `Layouts` e escolha a opção `Add layout`, como mostra a figura:

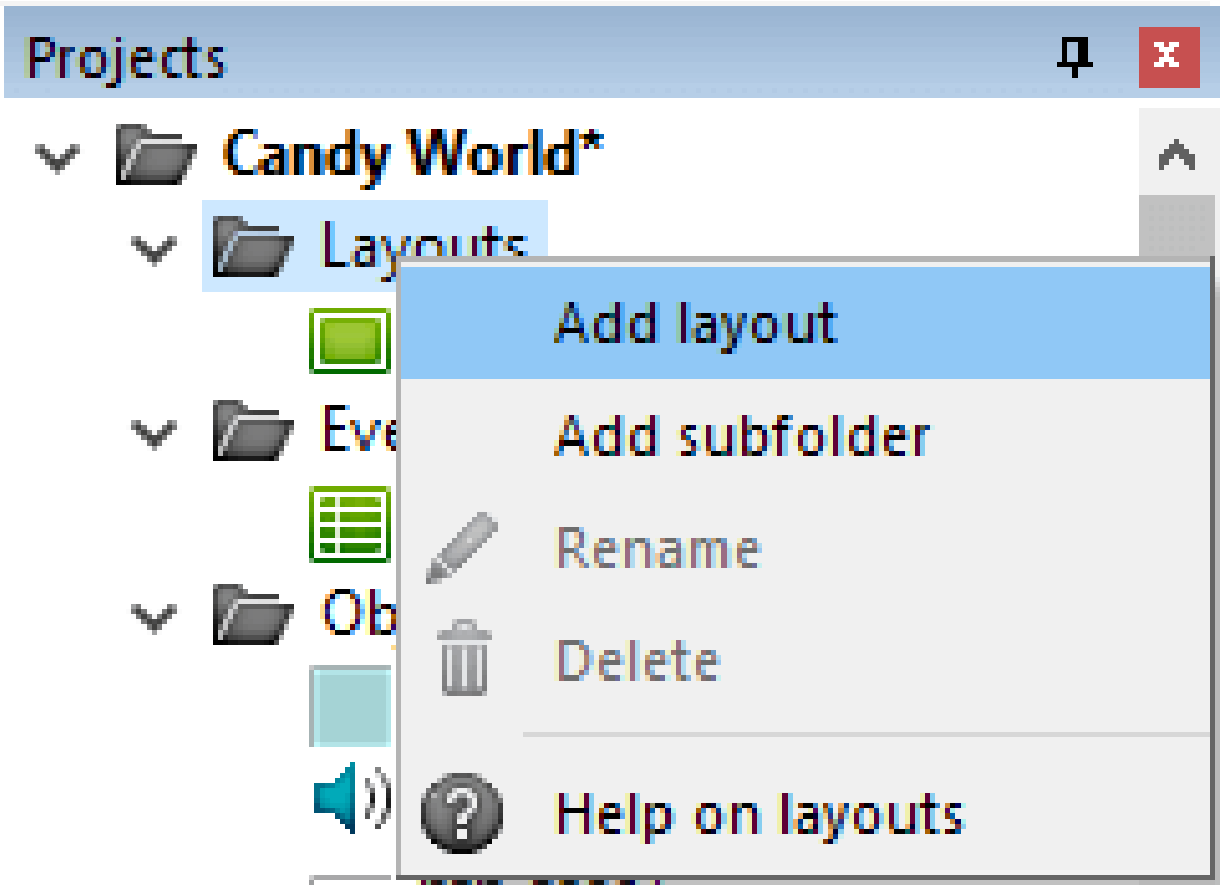


Figura 8.14: Add layout

Como não queremos criar uma nova folha de eventos, escolha a opção `Don´t add event sheet` (não adicionar folha de eventos) na janela `Add layout`, como a seguir:

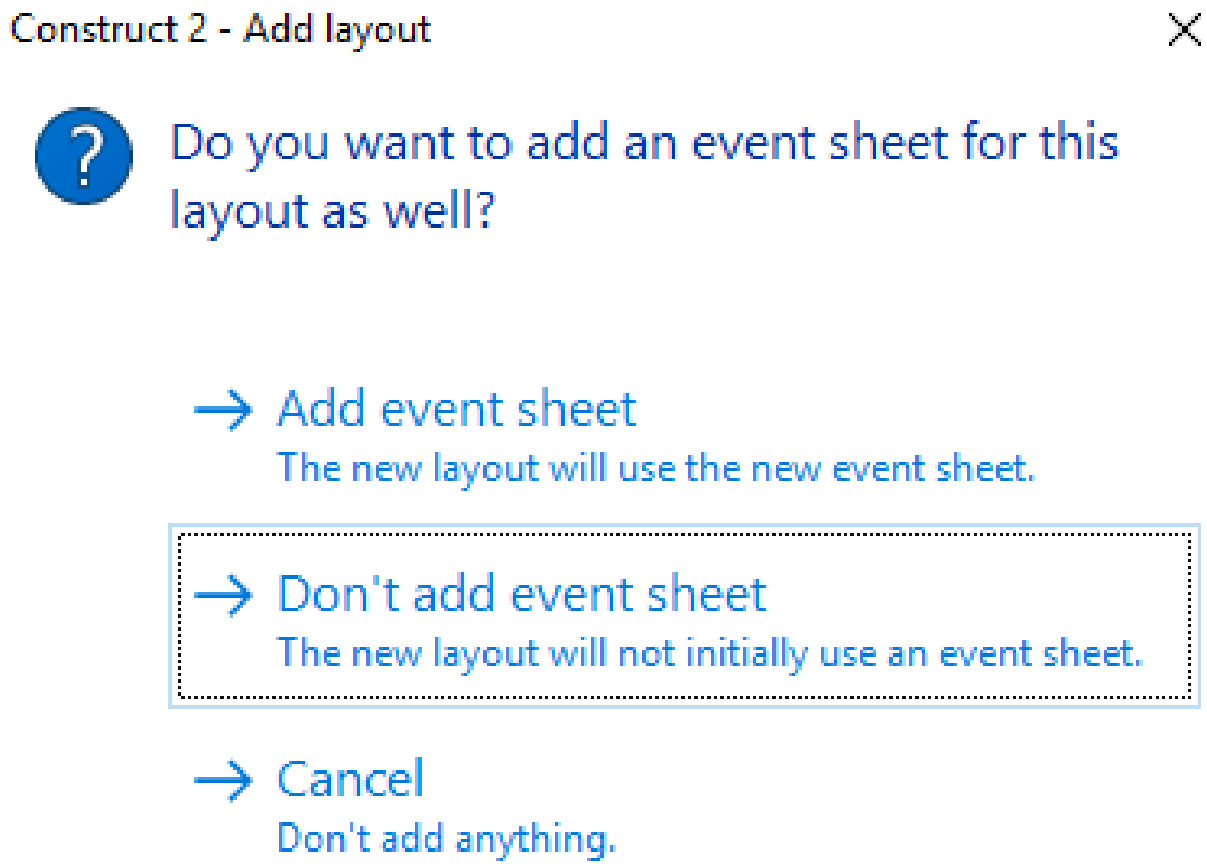

Construct 2 - Add layout
Do you want to add an event sheet for this layout as well?
Add event sheet
The new layout will use the new event sheet.
Don't add event sheet
The new layout will not initially use an event sheet.
Cancel
Don't add anything.

Figura 8.15: Don't add event sheet

Essa ação fará com que o novo layout continue vinculado à folha de eventos que já temos. Dê o nome `Telainicial` a ele e ajuste o seu tamanho no `Layout size` para 640x480 pixels. Escolhemos esse tamanho porque corresponde à parte da área pontilhada (ou seja, a área visível).

Importe um objeto de `Tiled background`, para ser a nossa imagem de fundo dessa tela. No editor de imagens, importe a imagem `telainicial` da nossa pasta `Telas`. Ajuste-a de modo que cubra toda a sua extensão, como mostra a figura seguinte:

Figura 8.16: Tela inicial do jogo

Por fim, como queremos que esta seja a primeira tela do jogo, precisamos dizer isso ao Construct 2. Portanto, vá até a aba de propriedades do projeto e procure pela opção `First layout`. Lá, procure pelo layout que acabamos de criar, o `Tela Inicial`:

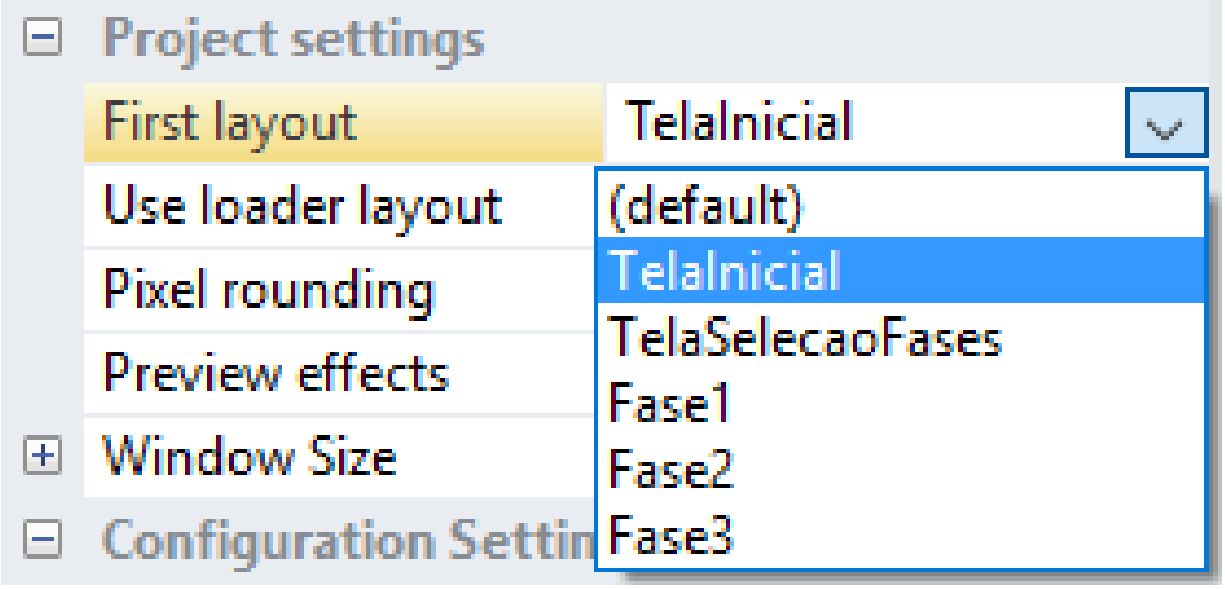


Figura 8.17: Escolha do primeiro layout

Seleção de fases

Vamos criar mais um layout, agora para ser a seleção de fases. Faça o mesmo processo anterior: adicione um novo layout e ajuste-o com o mesmo tamanho (640x480 pixels), pois também queremos somente a parte da área visível nele. Então, dê o nome `TelaSelecaoFases`.

Importe novamente um objeto de `Tiled background` e, no editor de imagens, importe a imagem `telaselecaofases`, que também está na pasta `Telas`. Ajuste para que cubra toda a extensão do layout, como fizemos com a `Telainicial`:

Figura 8.18: Tela de seleção de fases

8.4 Fase 2

Agora vamos criar mais uma fase para o nosso jogo. Para facilitar o nosso trabalho, vamos duplicar a `Fase1` e somente alterar o seu design, pois assim manteremos outros elementos que não mudam, como as layers, posição do jogador, tela de *game over* etc.

Para clonar um layout, selecione-o, clique com o botão direito e escolha a opção `Duplicate`, como mostra a figura:

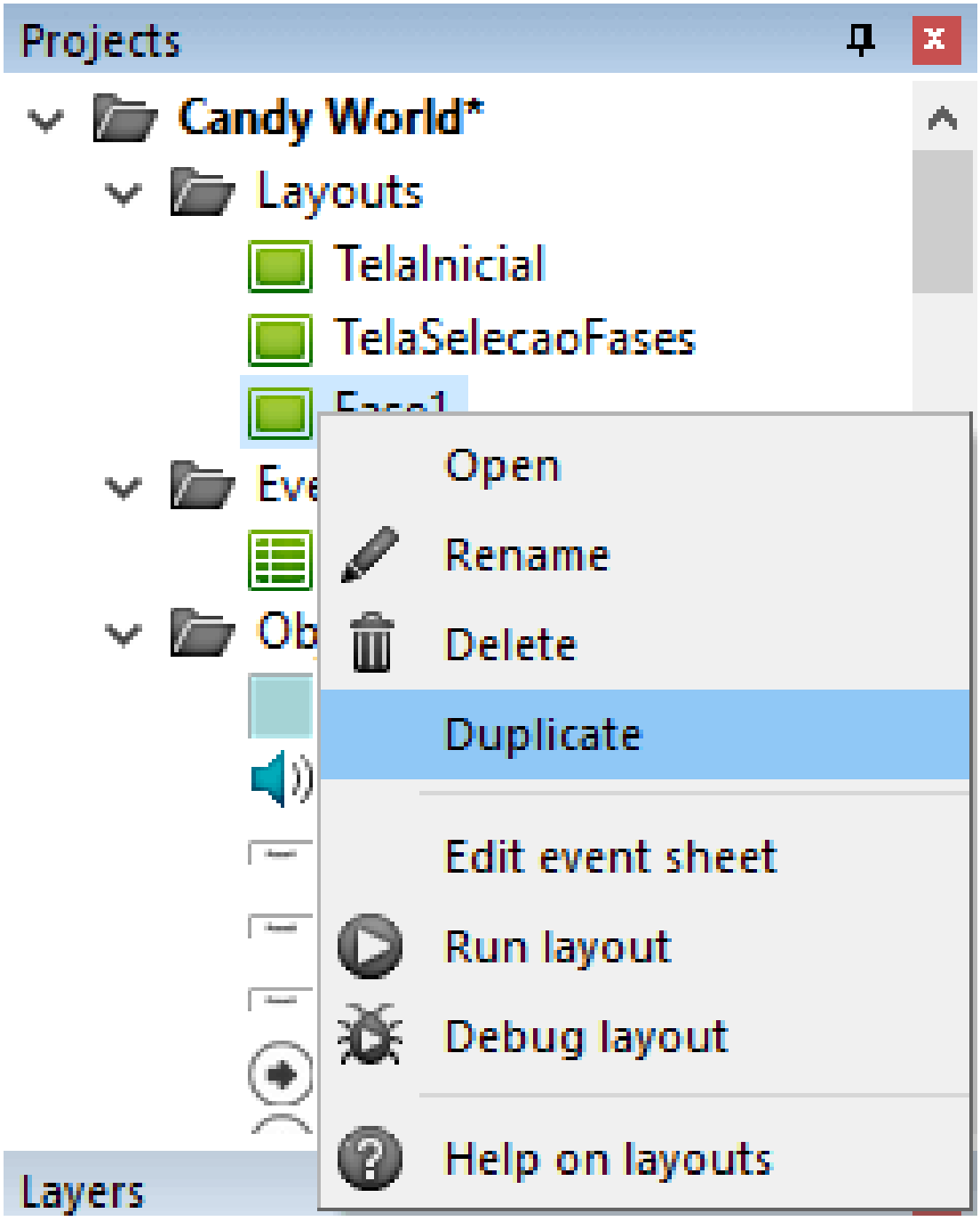


Figura 8.19: Duplicate layout

Dê o nome de `Fase2` nesse layout duplicado. A sequência dos layouts criados ficará assim:

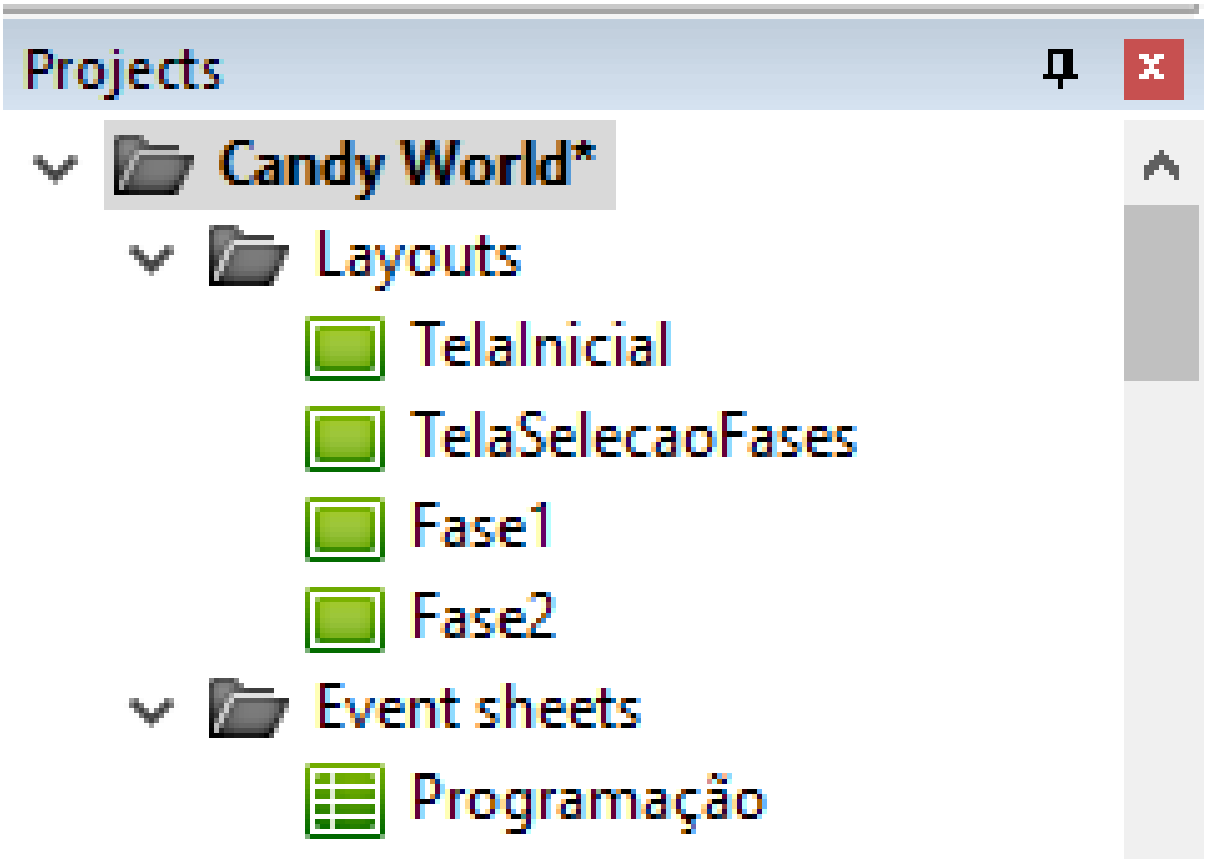


Figura 8.20: Sequência da barra de projetos

Com o layout criado, agora precisamos desenvolver um design diferente para a nossa segunda fase. Você pode colocar mais obstáculos e inimigos para ficar mais difícil do que a primeira. Utilize os diversos recursos da pasta disponibilizada e a sua criatividade para criá-la.

Agora que criamos todos os layouts que usaremos no jogo, vamos inserir mais uma ação para o evento criado de quando o jogador colide com a porta, incluído no capítulo anterior ao final da fase. Para essa ação, selecione o objeto de sistema (`System`), dê a ação de `Go to layout` e escolha a `TelaSelecaoFases` no campo `Layout` . O evento ficará desta forma:

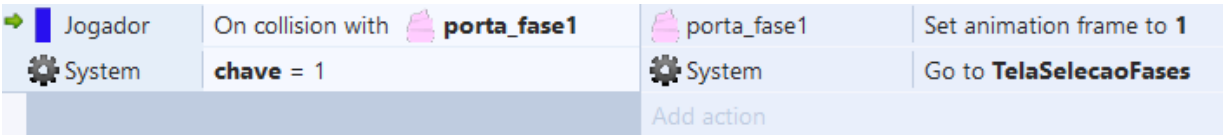


Figura 8.21: Evento de colisão com a porta

Portanto, toda vez que o jogador colidir com a porta e já estiver com a chave (variável `chave` igual a `1`), a porta abrirá (frame `porta` igual a `1`). Na última ação que incluímos agora, o jogador vai para a tela de seleção de fases.

No próximo capítulo, colocaremos os botões em todas as nossas telas.

Capítulo 9
Trabalhando com botões

Agora que temos todas as telas do jogo criadas, vamos começar a trabalhar com botões e finalizar alguns tópicos que já deixamos pré-prontos, como as telas de fim de jogo e de pausa. Veremos como criá-los e programá-los. O procedimento é bem simples. Ao final do capítulo, usaremos os conceitos para colocar o controle mobile com os devidos eventos para o jogo funcionar em qualquer dispositivo móvel.

9.1 Adicionando os botões

Vamos primeiramente adicionar todos os botões que precisaremos no jogo e, logo em seguida, começar a programar seus eventos. Começaremos pelo `Layout Telainicial`.

Insira o botão que será responsável por levar o jogador à tela de seleção de fases quando clicado, que será chamado de `Jogar`. Para isso, adicione uma `Sprite` com o nome `btn_jogar`, na camada `UI`. Importe nela a imagem `btn-jogar` da nossa pasta de recursos `Botões`, e posicione no layout, conforme mostramos a seguir:

Figura 9.1: Tela inicial com o botão começar inserido

Agora vamos inserir dois botões no `Layout TelaSelecaoFases` . O primeiro será responsável por levar o jogador para a primeira fase e o segundo, para a segunda fase do jogo. Para isso, crie duas `Sprites` na layer `UI` : uma com o nome `btn_fase1` , e outra com o nome `btn_fase2` . Em seguida, insira as duas imagens da nossa pasta de `Botões` chamadas `btn-fase1` e `btn-fase2` , respectivamente, nessas `Sprites` criadas. Posicione no layout conforme a seguir:

Figura 9.2: Tela de seleção de fases com os botões inseridos

Agora vamos partir para o botão de pausa que, quando clicado, vai pausar todo o sistema do jogo e mostrará a tela de pausa, inserida no capítulo *Telas de pausa e de fim de jogo*. Esse botão ficará posicionado ao lado superior direito, dentro da área pontilhada dos layouts `Fase1` e `Fase2`, de acordo com a figura a seguir:

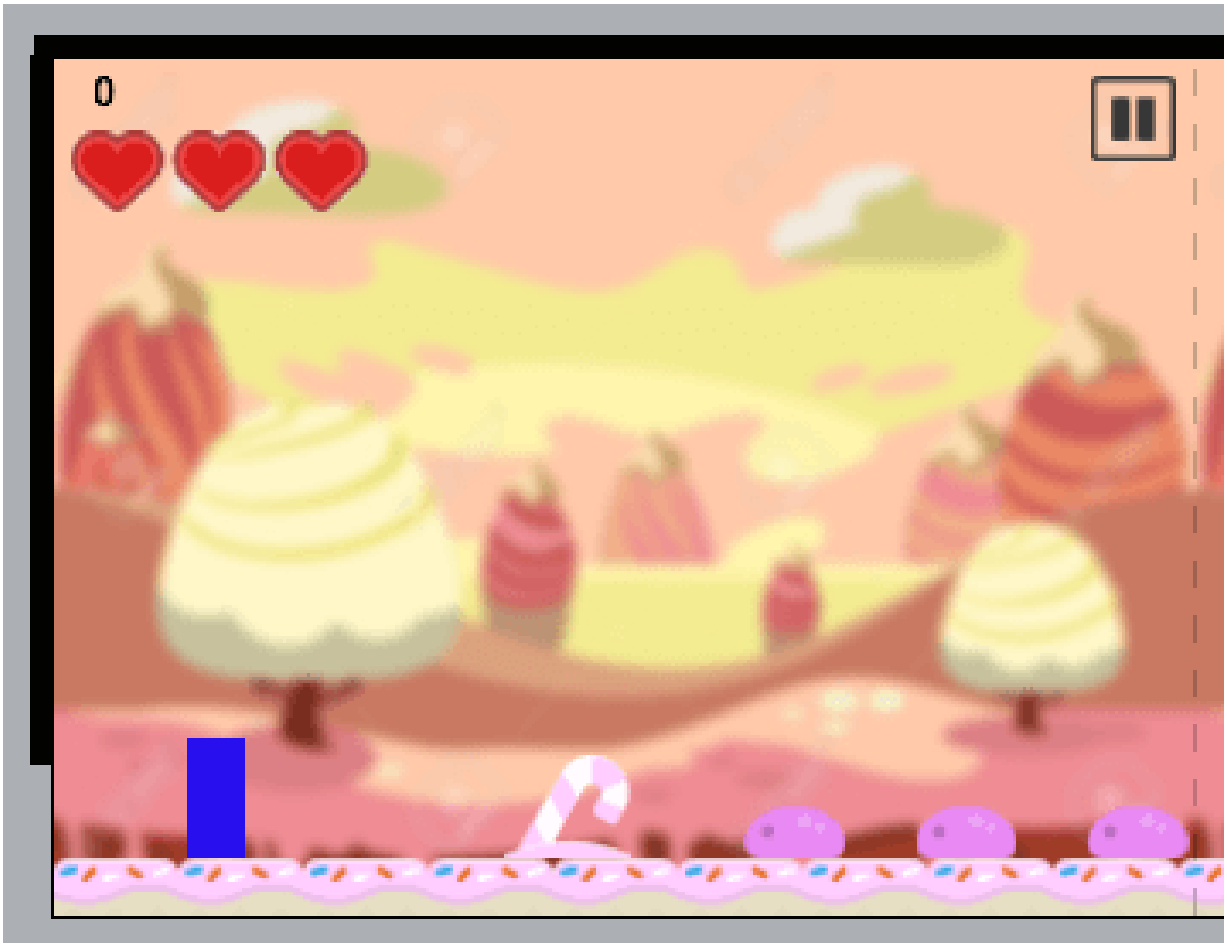


Figura 9.3: Botão de pausa inserido no layout

Para isso, adicione uma `Sprite` com o nome `btn_pausa` na layer `UI`, e importe nela a imagem `btn-pausa` da nossa pasta `Botões`.

Por último, vamos inserir dois botões para a tela de fim de jogo, e mais dois para a tela de pausa. Ambas terão um botão para voltar à tela de seleção de fases, entretanto, a de pausa terá um especialmente para voltar ao jogo enquanto que a de fim de jogo terá um que permitirá ao jogador tentar novamente aquela fase. Faça o mesmo processo que fizemos para os outros para inserir os botões nas `Sprites`, tudo sempre na layer `UI`.

Para a tela de fim de jogo, importe os botões `btn-positivo` e `btn-negativo`, e dê o nome de `btn_telaGameoverSim` e `btn_telaGameoverNao` nas `Sprites`, respectivamente. Já para a tela de pausa, importe os

botões `btn-continue` e `btn-fases`, e dê o nome de `btn_telaPauseContinue` e `btn_telaPauseFases`, respectivamente.

Lembre-se de que todos eles estão disponibilizados na nossa pasta de recursos, a `Botões`. Posicione-os exatamente como mostra a figura a seguir.

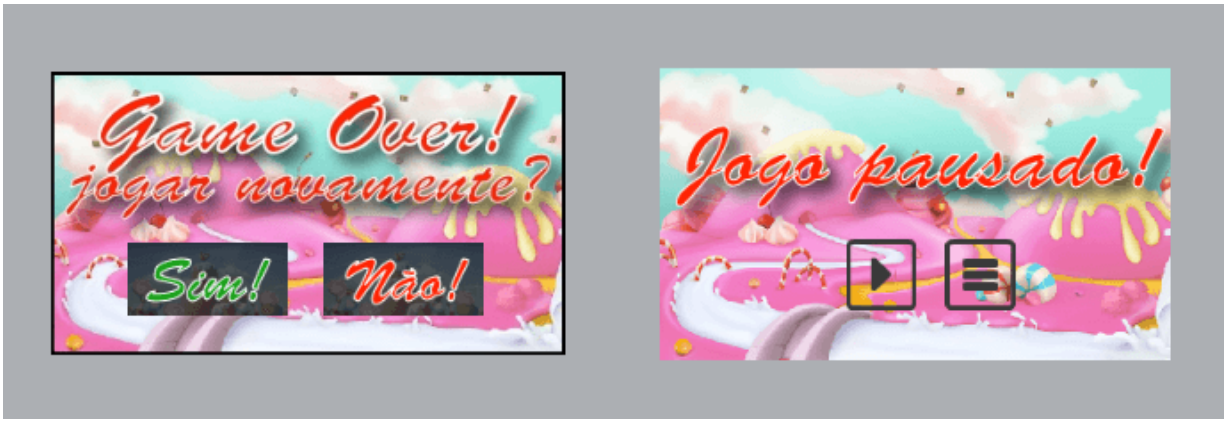


Figura 9.4: Telas de fim de jogo e de pausa com os botões inseridos

Adicione uma `Behavior Pin` nos quatro botões importados agora. Assim, poderemos programá-los para que fiquem "colados" às telas de *game over* e de pausa quando elas "descerem" para o meio do layout.

Pronto! Acabamos de adicionar os comportamentos necessários em todos os botões do nosso jogo. Muito fácil, não é mesmo? O próximo passo é adicionar os eventos referentes a eles. Vamos lá!

9.2 Programando os eventos dos botões

Na folha de eventos, colocaremos mais quatro ações na condição `On start of layout`: duas serão dos botões da tela de fim de jogo (`btn_telaGameoverSim` e `btn_telaGameoverNao`) com a ação `Pin to object` no objeto `Tela_gameover`; e mais duas dos botões da tela de pausa

(`btn_telaPauseContinue` e `btn_telaPauseFases`) com a ação `Pin to object` no objeto `Tela_pause` .

Isso fará com que todos os botões fiquem "colados" em suas respectivas telas toda vez que o layout for iniciado. O nosso evento `On start of layout` ficará como a seguir:

Figura 9.5: Evento On start of layout

Eventos da tela de pausa

Agora vamos inserir os eventos com as ações para cada botão que criamos. Para o botão `btn_pausa` , adicionaremos as que farão com que o sistema do nosso jogo "pare" toda vez que for clicado.

Primeiro, vamos fazer com que nosso jogador fique desativado quando o jogo estiver pausado. Para isso, escolha o objeto `Touch` , dê a condição `Is touching object` e escolha o objeto `btn_pausa` . Na ação desse evento, selecione o objeto `Jogador` e dê a ação `Set enable` (definir habilitação) e, no campo `State` (estado), escolha `Disabled` (desativado), como mostra a figura a seguir.

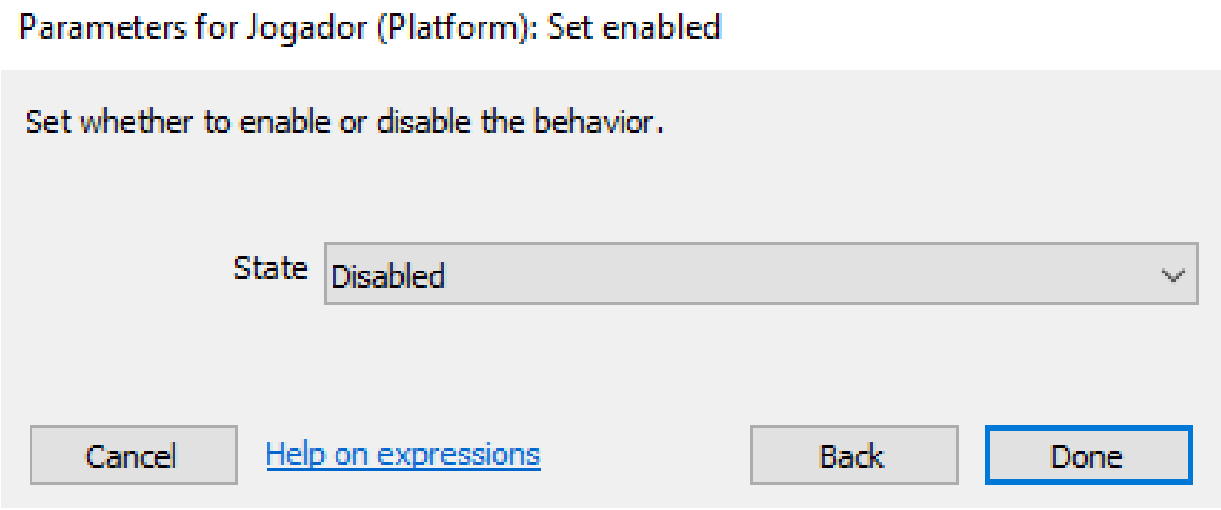


Figura 9.6: Set enable

Na próxima ação, faremos com que o tempo do layout "pare" também quando o jogo estiver pausado. Para isso, escolha o objeto de sistema (`System`) e dê a ação `Set time scale` . No campo `Time scale` , coloque o valor `0` (zero).

Para a próxima e última ação, vamos fazer com que a tela de pausa apareça no meio do layout quando o jogo estiver pausado. Para isso, será necessário mudar a posição da tela para as coordenadas que correspondem a exatamente o meio da tela da fase. Escolha o objeto `Tela_pause` , dê a ação `Set position` , e coloque os valores 320 e 240 para X e Y, respectivamente. No final, nosso evento ficará como a figura a seguir.

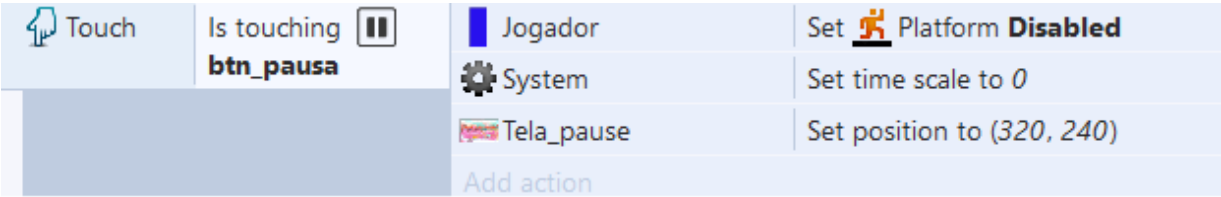


Figura 9.7: Evento do botão de pausa

Agora, vamos adicionar os eventos referentes aos dois botões contidos na tela de pausa. Queremos que o `btn_telaPausaContinue` despause o jogo e faça com que nosso jogador continue

exatamente de onde parou. Para isso, precisamos fazer três ações contrárias às ações que fizemos para pausar o jogo.

Para esse evento, vamos escolher o objeto `Touch` e dar a condição `On touched object`. Escolha o objeto `btn_telaPausaContinue`. Na ação desse evento, selecione o objeto `Jogador` e dê a ação `Set enable`. No campo `State`, escolha `Enable`. Essa ação fará com que nosso personagem seja destravado.

Na próxima ação, vamos voltar ao tempo normal do layout. Para isso, escolha o objeto de sistema (`System`) e dê a ação `Set time scale`. No campo `Time scale`, coloque o valor `1`.

Na próxima e última ação, faremos com que a tela de pausa desapareça do meio da tela. Logo, teremos de definir uma posição totalmente fora do layout. Vamos escolher o objeto `Tela_pause` e dar a ação `Set position`. Coloque os valores 320 e -4000 para X e Y, respectivamente. Eles são os valores para a tela de pausa ir a um ponto totalmente fora do layout.

No final, nosso evento ficará assim:

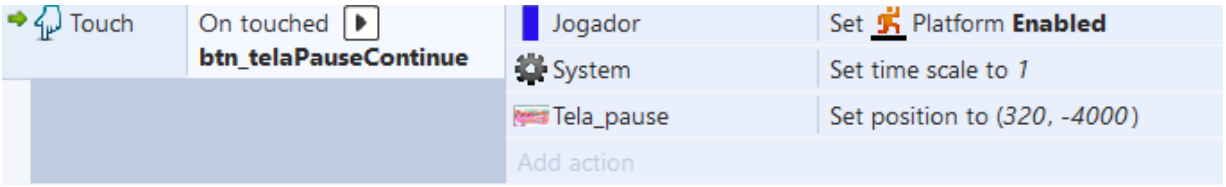


Figura 9.8: Evento do botão continuar

Agora vamos adicionar o evento do botão `btn_telaPausaFases`. Queremos que ele seja responsável por levar o jogador à tela de seleção de fases. Então, vamos escolher o objeto `Touch` e dar a condição `On touched object`. Depois, selecione o objeto `btn_telaPausaFases`.

Na primeira ação desse evento, vamos de novo voltar o tempo normal do layout. Precisamos fazer isso para evitar que o jogo continue congelado mesmo depois de sair da fase. Escolha o objeto

de sistema (`System`), dê a ação `Set time scale` e, no campo `Time scale` , coloque o valor `1` . Lembre-se de que o `1` representa a velocidade padrão.

Na próxima ação, escolha novamente o objeto `System` e dê a ação `Reset global variables` . Isso fará com que todas as variáveis globais sejam reiniciadas quando sair do layout, assim, quando começar um novo jogo, as variáveis não estarão guardando valores do anterior.

Na última ação desse evento, vamos fazer com que o jogador vá à tela de seleção de fases. Logo, escolha o objeto `System` mais uma vez e dê a ação `Go to layout` . No campo `Layout` , escolha o `TelaSelacaoFases` , como mostra a seguir.

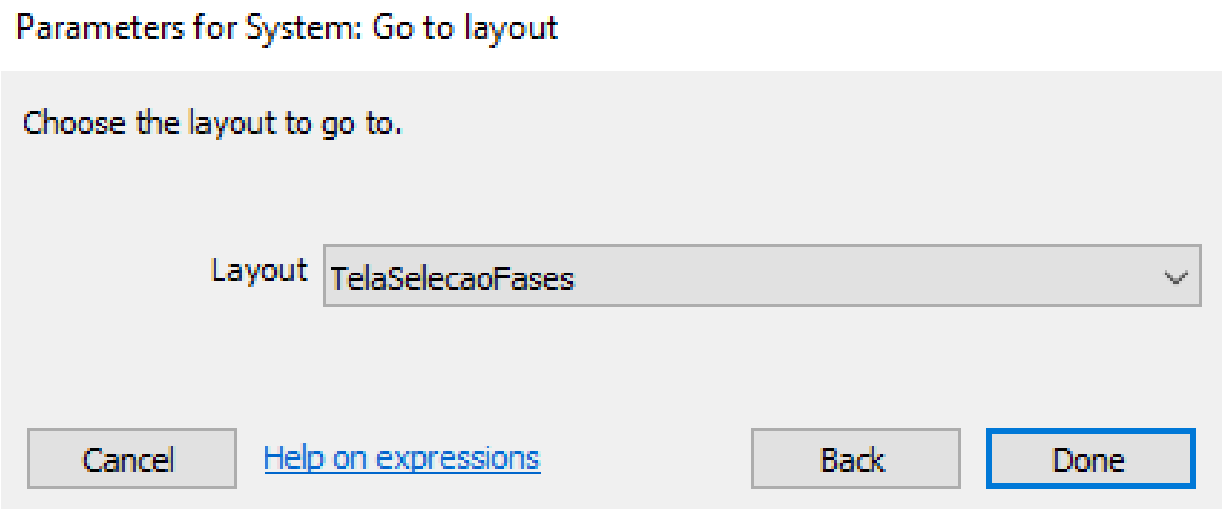


Figura 9.9: Go to layout

Por fim, o nosso evento ficará como a figura seguinte:

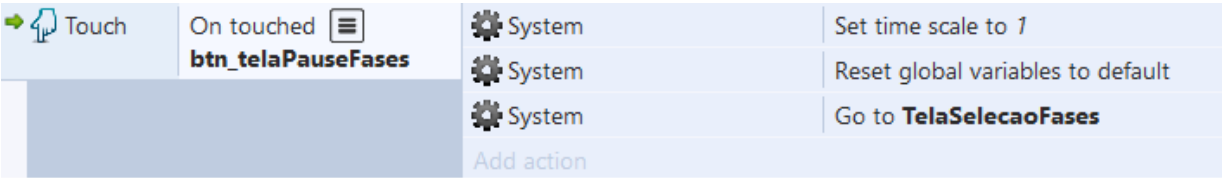


Figura 9.10: Evento do botão de fases

Pronto, agora o nosso sistema de pausa está todo programado! Teste e veja como os botões estão funcionando corretamente.

Eventos da tela de fim de jogo

O próximo passo é inserir os eventos dos dois botões da nossa tela de fim de jogo. Para o evento do botão `btn_telaGameoverSim`, responsável por fazer o jogador continuar a jogar, vamos escolher o objeto `Touch`, dar a condição `On touched object` e selecionar o objeto `btn_telaGameoverSim`.

Na primeira ação desse evento, queremos ativar o comportamento de plataforma do nosso jogador. Para isso, escolha o objeto `Jogador`, dê a ação `Set enable` e, no campo `State`, escolha `Enable`.

Na segunda ação, novamente voltaremos ao tempo normal do nosso layout. Como fizemos das outras vezes, escolha o objeto de sistema (`System`), dê a ação `Set time scale` e, no campo `Time scale`, coloque o valor `1`.

Na terceira ação, escolha o objeto `System` e dê a ação `Reset global variables`. Isso fará com que todas as variáveis globais do nosso jogo sejam iniciadas.

Para a próxima e última ação desse evento, escolha o objeto `System` e dê a ação `Restart layout`. Assim, o layout será reiniciado e o jogo começará do início. Ao final, nosso evento ficará assim:

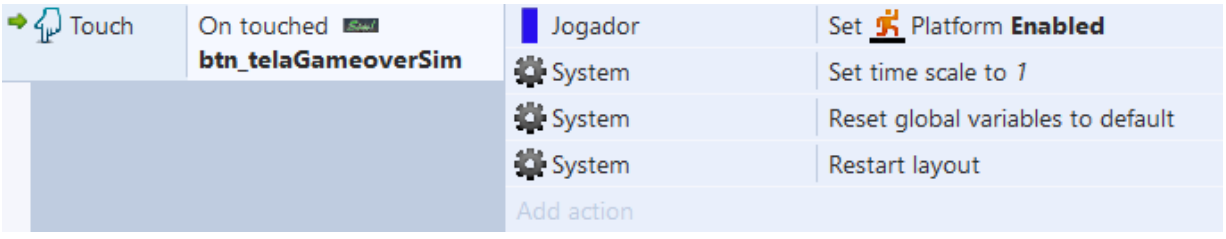


Figura 9.11: Evento do botão telaGameoverSim

Para o evento do botão `btn_telaGameoverNao`, responsável por fazer o jogador voltar à tela inicial do jogo, vamos escolher o objeto `Touch` e

dar a condição `On touched object`. Depois, escolha o objeto `btn_telaGameoverNao`.

Para a primeira ação desse evento, vamos fazer com que o comportamento de plataforma do `Jogador` seja desativado. Faremos isso para evitar que o jogador consiga movimentar o personagem quando ele está sem vidas. Para isso, escolha o objeto `Jogador`, dê a ação `Set enable` e, no campo `State`, escolha `Disabled`.

Para a segunda ação, vamos mais uma vez voltar ao tempo normal do layout. Então, escolha o objeto de sistema (`System`), dê a ação `Set time scale` e, no campo `Time scale`, coloque o valor `1`. Para a terceira ação, escolha o objeto `System` e dê a ação `Reset global variables`, para que as variáveis globais sejam iniciadas.

Na quarta e última ação deste evento, faremos com que o jogador vá à tela inicial do jogo. Logo, escolha o objeto `System`, dê a ação `Go to layout` e, no campo `Layout`, selecione o `TelaInicial`. Nosso evento ficará como a figura a seguir.

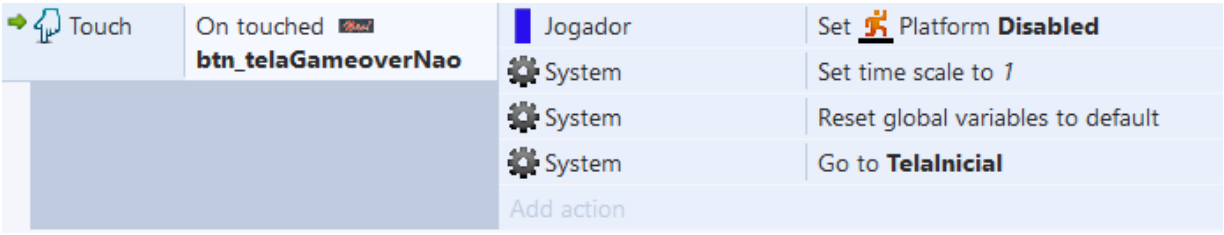


Figura 9.12: Evento do botão telaGameoverNao

Pronto, terminamos o sistema da nossa tela de fim de jogo. Agora só falta adicionarmos os eventos dos botões da tela de seleção de fases e do botão `Jogar` da tela inicial.

Eventos da tela de seleção de fases

Para os botões `btn_fase1` e `btn_fase2` da tela de seleção de fases, vamos adicionar um evento para quando um ou outro for clicado, o jogador irá para a primeira ou segunda fase do nosso jogo, respectivamente.

Assim, vamos escolher o objeto `Touch`, dar a condição `On touched object`, e selecionar o objeto `btn_fase1`. Na ação, escolha o objeto `System`, dê a ação `Go to layout` e, no campo `Layout`, escolha a `Fase1`. Faça a mesma coisa com o objeto `btn_fase2`; a diferença é que, no campo `Layout` da ação `Go to layout`, você precisa escolher a `Fase2`.

Eventos da tela inicial

Por fim, vamos adicionar o evento do botão `Jogar` da tela inicial. Quando clicado, ele levará o jogador à tela de seleção de fases.

Desta forma, escolha o objeto `Touch`, dê a condição `On touched object`, e selecione o objeto `btn_jogar`. Para a ação, escolha o objeto `System`, dê a ação `Go to layout` e, no campo `Layout`, escolha a `TelaSelecaoFases`.

Esses três últimos eventos ficarão como a seguir:

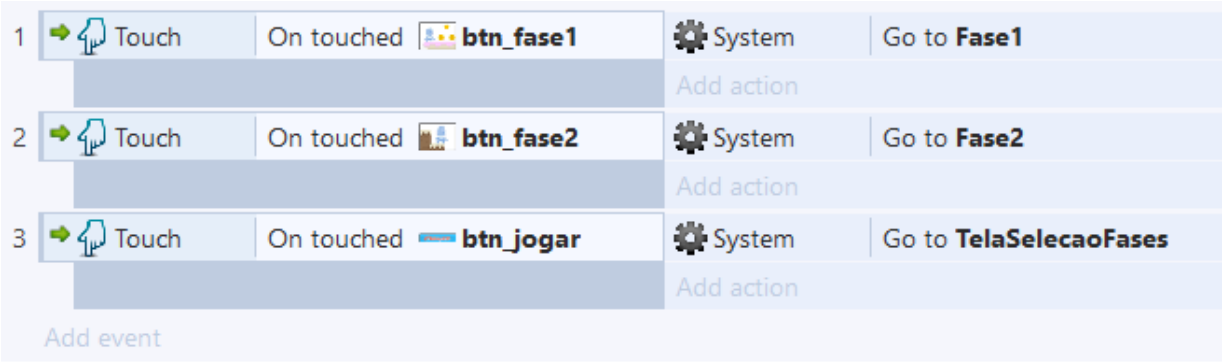

1 Touch On touched btn_fase1 System Go to Fase1
Add action
2 Touch On touched btn_fase2 System Go to Fase2
Add action
3 Touch On touched btn_jogar System Go to TelaSelecaoFases
Add action
Add event

Figura 9.13: Evento dos botões da tela inicial e de seleção de fases

9.3 Adicionando e programando os controles mobile

Neste momento, precisamos inserir os três botões que farão o jogador pular e andar para a direita e a esquerda nos dispositivos

móveis. Assim, importe uma `Sprite` com o nome de `btn_mobile_Esquerda` na layer `UI`, e insira nela a imagem `btn-mobile-esquerda` da nossa pasta de `Botões`. Importe uma outra `Sprite` na layer `UI` com o nome de `btn_mobile_Direita`, e insira nela a imagem `btn-mobile-direita` da mesma pasta. Por fim, importe mais uma `Sprite` com o nome de `btn_mobile_Pulo` na layer `UI`, e insira a imagem `btn-mobile-pulo` nela.

Adicione uma colisão simples (`Set to bounding box`), como explicamos no capítulo *Estrutura inicial*, nessas três imagens importadas e posicione nos layouts `Fase1` e `Fase2`, como mostra a seguir.

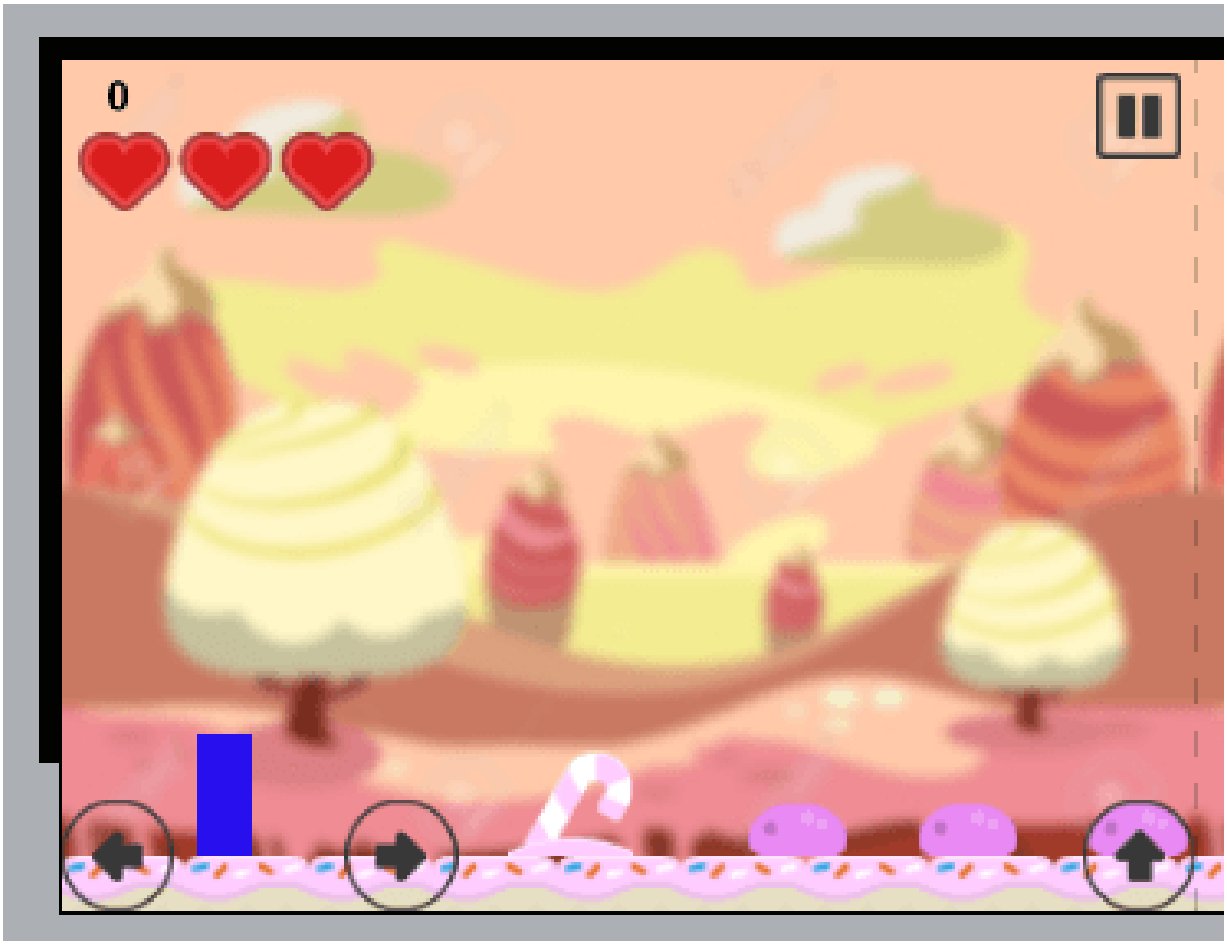


Figura 9.14: Botões mobile inseridos no layout

Agora vamos adicionar os eventos a esses botões. Para o botão `btn_mobile_Esquerda`, responsável por fazer o `Jogador` se movimentar para a esquerda, vamos adicionar o seguinte evento. Escolha o objeto `Touch`, dê a condição `Is touching object`, e selecione o objeto `btn_mobile_Esquerda`.

Para a ação, escolha a `Sprite Jogador`, dê a ação `Simulate control` e, no campo `Control`, escolha a opção `Left`. Ela fará com que o comportamento de plataforma do jogador seja simulado para a esquerda. Então, a cada toque neste botão, ele dará um passo à esquerda e, para fazê-lo andar, é necessário manter este botão pressionado.

Na próxima ação, escolha a `Sprite JogadorAnimacao`, dê a ação `Set mirrored` e, no campo `State`, escolha a opção `Mirrored`. Essa ação fará com que a animação do jogador seja setada para o lado que ele estiver andando.

Para o botão `btn_mobile_Direita`, responsável por fazer o `Jogador` se movimentar para a direita, vamos adicionar o seguinte evento. Escolha o objeto `Touch`, dê a condição `Is touching object`, e escolha o objeto `btn_mobile_Direita`. Para a ação, escolha a `Sprite Jogador`, dê a ação `Simulate control` e, no campo `Control`, escolha a opção `Right`. Ela fará com que o comportamento de plataforma do jogador seja simulado para a direita.

Na próxima ação, escolha a `Sprite JogadorAnimacao`, dê a ação `Set mirrored` e, no campo `State`, escolha a opção `Not Mirrored`. Isso fará com que a animação do jogador seja setada para o lado que ele andar – ou seja, para a direita nesse caso. Com isso, teremos os seguintes eventos:

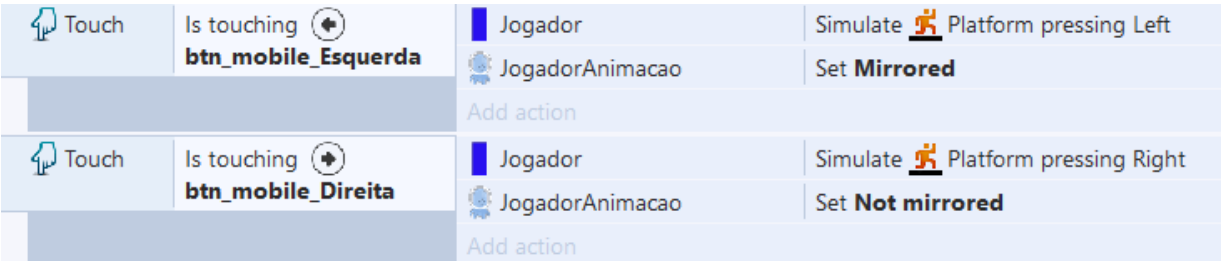


Figura 9.15: Eventos dos botões mobile dos movimentos de esquerda e direita

Por último, para o botão `btn_mobile_Pulo`, que será responsável por fazer o `Jogador` pular, vamos adicionar o seguinte evento. Escolha o objeto `Touch`, dê a condição `Is touching object` e escolha o objeto `btn_mobile_Pulo`.

Para a ação, escolha a `Sprite Jogador`, dê a ação `Simulate control` e, no campo `Control`, escolha a opção `Jump` (pular), como mostra a figura a seguir. É possível alterar a força do pulo nas propriedades da `Behavior Platform` do objeto `Jogador`:

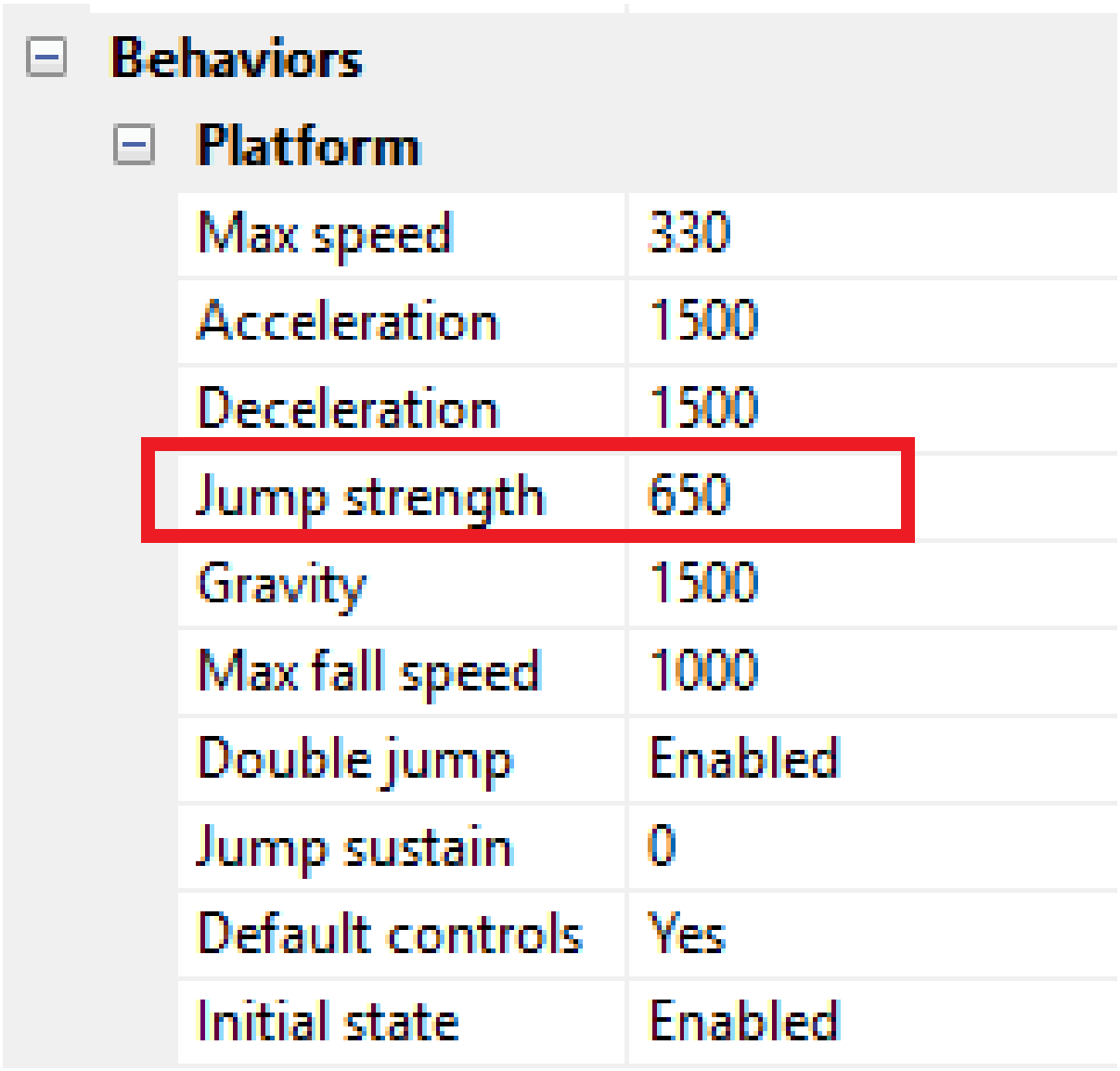


Figura 9.16: Configurando a altura do pulo do jogador

Para o nosso jogo, mantemos a configuração padrão de 650 pixel por segundo, porém, dependendo da altura que você colocar os obstáculos/plataformas no seu jogo, você pode modificar o campo `Jump Strength` (Força do pulo) para um valor mais alto ou mais baixo.

Figura 9.17: Evento dos botões mobile do movimento de pulo

Pronto, agora temos todos os nossos botões funcionando corretamente. Já podemos testar e até jogar para ver como está ficando! No próximo capítulo, ensinaremos como dar mais vida ao jogo ao inserir efeitos sonoros e música de fundo.

Capítulo 10
Música e sons

Há vários aspectos que precisam ser considerados quando criamos um jogo, e passamos por muitos deles ao longo dos capítulos. Falamos sobre jogabilidade, usabilidade, gráficos etc., mas ficou faltando algo, não é mesmo? Neste capítulo, explicaremos tudo o que é necessário saber para trabalhar com áudio dentro do Construct 2.

Mostraremos como adicionar arquivos de áudio na plataforma, quais formatos são aceitos, quais são as diferenças entre os efeitos especiais e as músicas de fundo, e como podemos colocá-los para tocar de acordo com os eventos do jogo. Não se preocupe, é tudo extremamente simples. Preparado? Vamos lá.

10.1 Arquivos de áudio no Construct 2

Nesta seção, vamos ensinar como importar arquivos de áudio no nosso jogo. Mas primeiro precisamos entender a diferenciação que o Construct 2 faz entre dois tipos do áudio, sendo que cada um tem funções distintas. Estamos falando dos `Sounds` (sons) e `Music` (música), arquivos de áudio usados para efeitos de som e música de fundo, respectivamente.

Usamos os sons para efeitos sonoros de curta duração, que são reproduzidos quando acontecem eventos programados pelo usuário, como uma colisão ou explosão. Já a música é utilizada em trilhas musicais mais longas. É o áudio que vai acompanhar todo o caminho do jogador, independente do que ele estiver fazendo.

O Construct 2 faz a diferenciação dos áudios na estrutura do nosso projeto. Se voltarmos um pouco a nossa atenção para a árvore de

diretórios do projeto, veremos que não há uma pasta chamada `Áudios`, mas existem duas: `Music` e `Sounds`, como mostra a figura a seguir. Esta organização não é somente para separar visualmente os arquivos. O Construct 2 também os trata de forma diferente internamente.

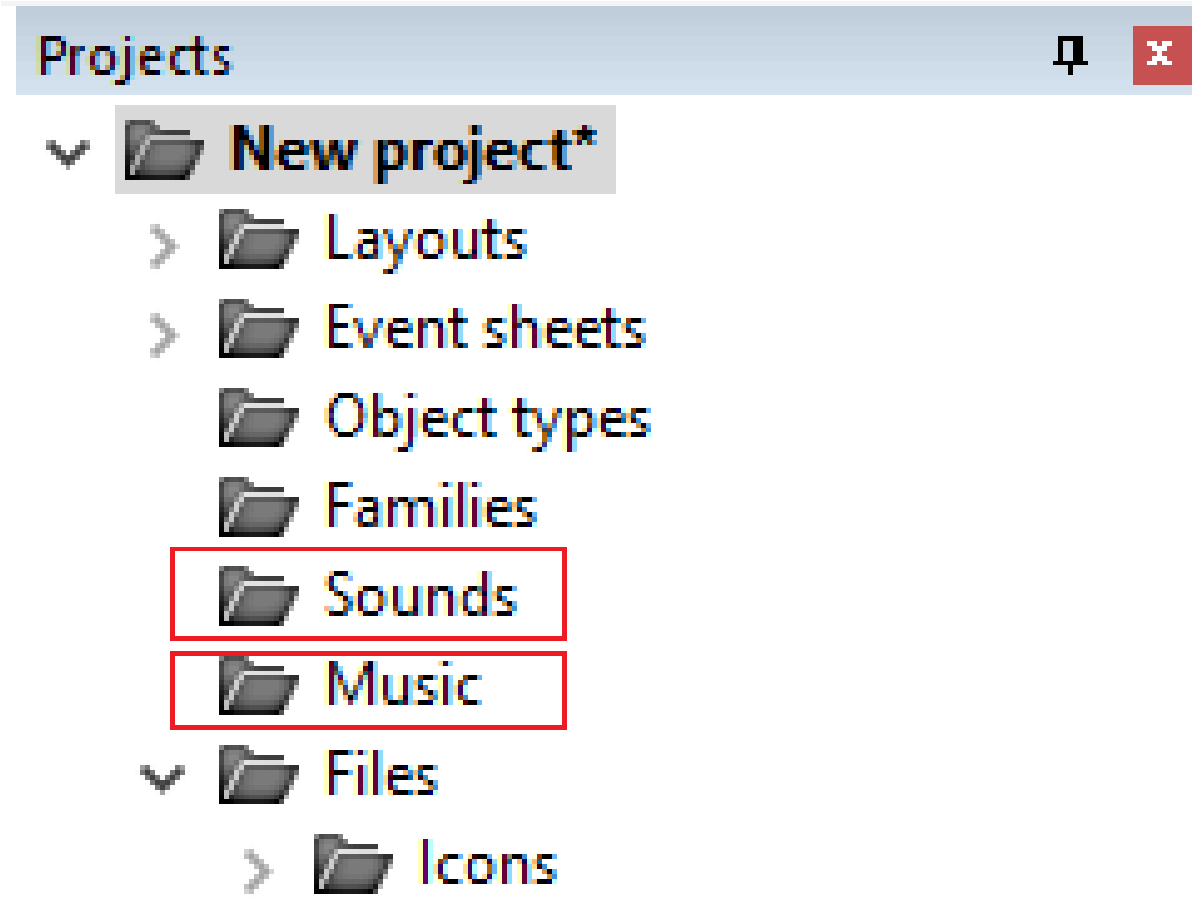


Figura 10.1: Pastas de áudios

Os sons que são utilizados para efeitos são todos baixados antes de serem reproduzidos, enquanto as músicas são baixadas de acordo com sua transmissão. Isso quer dizer que, se uma faixa de música qualquer for colocada erroneamente na pasta de `Sounds`, ela será carregada totalmente antes de ser reproduzida, ocasionando atrasos no jogo. Do mesmo modo, se um som de efeito for colocado

na pasta de `Music`, ele será reproduzido imediatamente, o que pode ocasionar efeitos estranhos.

Todos os arquivos de áudio do projeto, sejam áudios ou músicas, devem ser importados nos formatos Ogg Vorbis (`.ogg`) e MPEG-4 AAC (`.m4a`). O Construct 2 exige os dois formatos para o mesmo áudio para conseguir dar suporte a todos os navegadores e devices. Mas para facilitar a nossa vida, podemos importar o áudio somente no formato `.wav`, que o Construct vai convertê-lo automaticamente para os dois formatos necessários.

Agora que entendemos o papel dos dois tipos de áudio e quais são suas diferenças, vamos colocar a mão na massa.

10.2 Adicionando os efeitos sonoros

Na pasta `Áudios` que está dentro da nossa pasta de recursos, há uma melodia que colocaremos como música de fundo no layout da `Fase1` e da `Fase2`. Lá também existem quatro efeitos sonoros que vamos acrescentar a algumas situações do jogo, como quando o jogador "destruir" um inimigo, "recolher" um item coletável ou até clicar em um botão. Para facilitar, todos os arquivos de áudios disponibilizados já estão em formato `.wav`.

Começaremos trabalhando com os efeitos sonoros. Encontre os arquivos `som-botao`, `som-chave`, `som-estrela` e `som-inimigo`, e depois os importe para a plataforma. Para adicionar um som, clique com o botão direito do mouse sobre a pasta `Sounds` e na opção `import sounds` (importar sons), como mostra a figura a seguir:

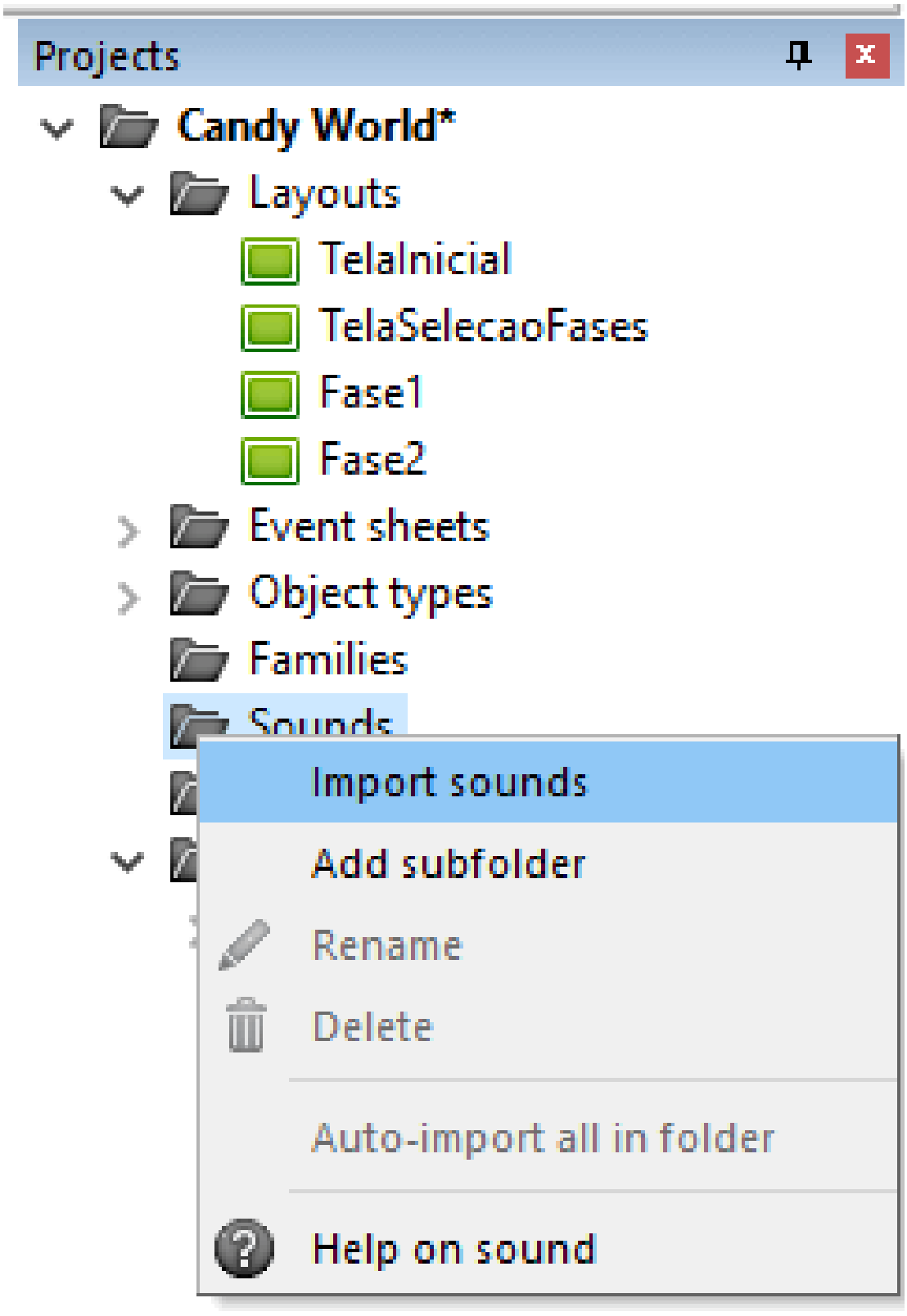


Figura 10.2: Importando um arquivo de som

Na janela `Import audio files` (importar arquivos de áudio), importe o áudio chamado `som-botao` e, em seguida, clique em `Import` :

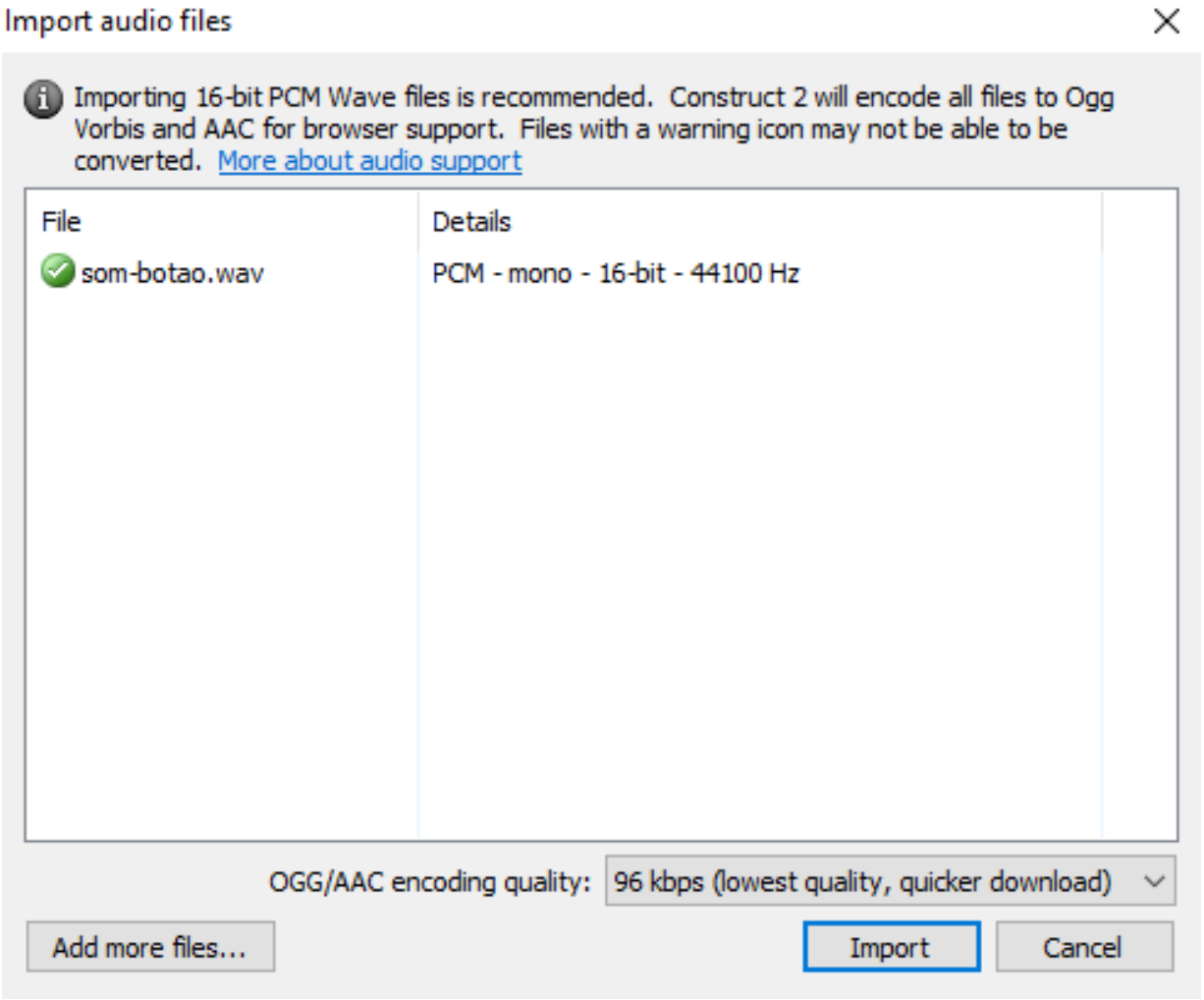

Import audio files
Importing 16-bit PCM Wave files is recommended. Construct 2 will encode all files to Ogg Vorbis and AAC for browser support. Files with a warning icon may not be able to be converted. More about audio support
File
Details
som-botao.wav
PCM - mono - 16-bit - 44100 Hz
OGG/AAC encoding quality:
96 kbps (lowest quality, quicker download)
Add more files...
Import
Cancel

Figura 10.3: Import audio files

Note que o Construct 2 vai automaticamente converter para dois arquivos de áudio nos formatos `.ogg` e `.m4a` , dentro da pasta `Sounds` , como mostra a figura:

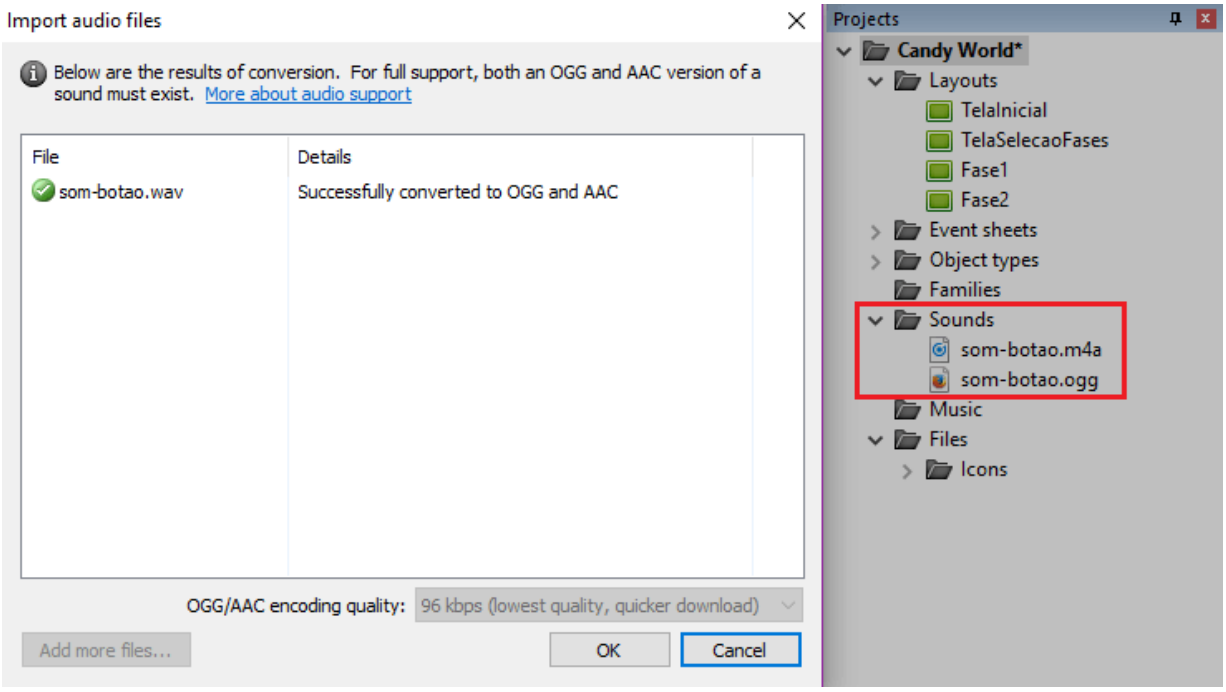


Figura 10.4: Som importado com sucesso

Pronto, o efeito sonoro `som-botao` que usaremos quando o jogador clicar em algum botão já está importado. Faça o mesmo processo para importar os outros três sons da nossa pasta `Áudio`.

Feito isso, precisaremos de um objeto que seja responsável por fazer toda a lógica de sons dentro do nosso jogo. Esse objeto é o `Audio`. Uma vez importado em qualquer layout, este objeto se torna global, sendo acessível em qualquer folha de estilos. É o mesmo que acontece que os objetos `Touch` e `Keyboard` que usamos anteriormente.

Na aba `Insert a new object` importe o objeto `Áudio` para dentro do layout, como mostra a figura:

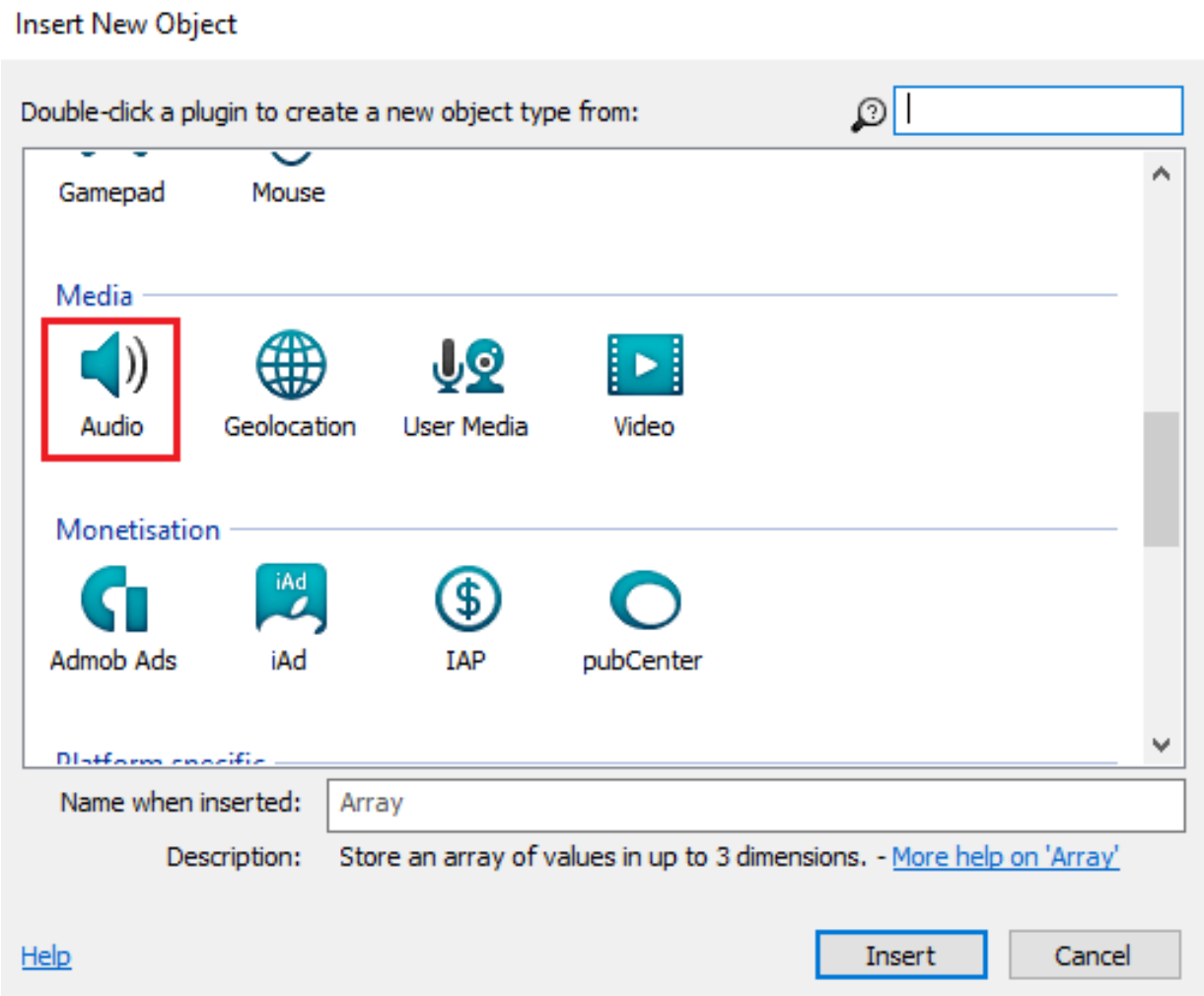


Figura 10.5: Importando um objeto de áudio

Maravilha. Agora basta associar os eventos do jogo aos sons que importamos. Para isso, vamos voltar alguns passos.

Você se lembra dos eventos de `Touch` que criamos para quando o jogador clicar nos botões `btn_fase1`, `btn_fase2` e `btn_jogar`? Adicionaremos neles mais uma ação. Escolha o objeto de `Audio` e dê a ação `Play`. No campo `Audio file` da janela de parâmetros, escolha o `som-botao`, coloque o nome da tag de `botao` e clique em `Done`, como a seguinte figura:

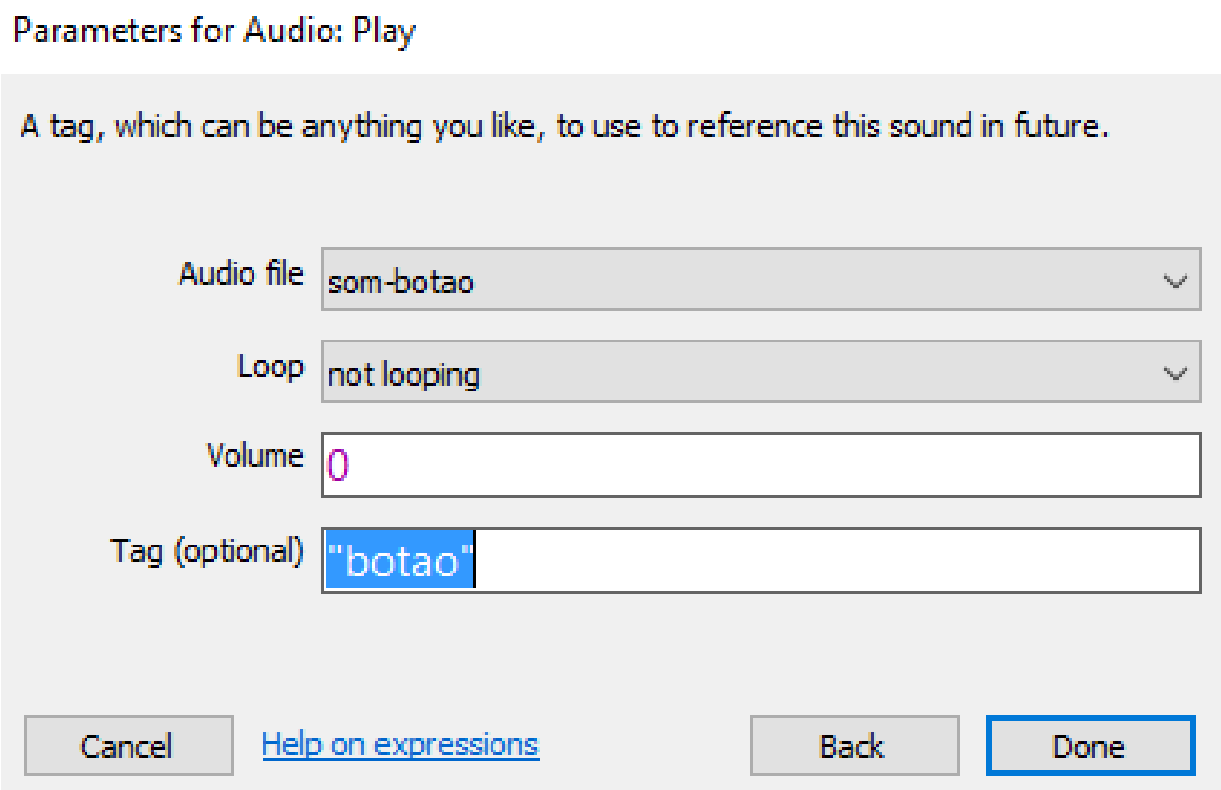


Figura 10.6: Parâmetros do objeto Audio

Isso fará com que o `som-botao` toque quando o jogador clicar nos botões: `btn_fase1`, `btn_fase2` e `btn_jogar`. Faça exatamente o mesmo processo para os outros botões citados. Ao final, nossa folha de eventos estará dessa forma:

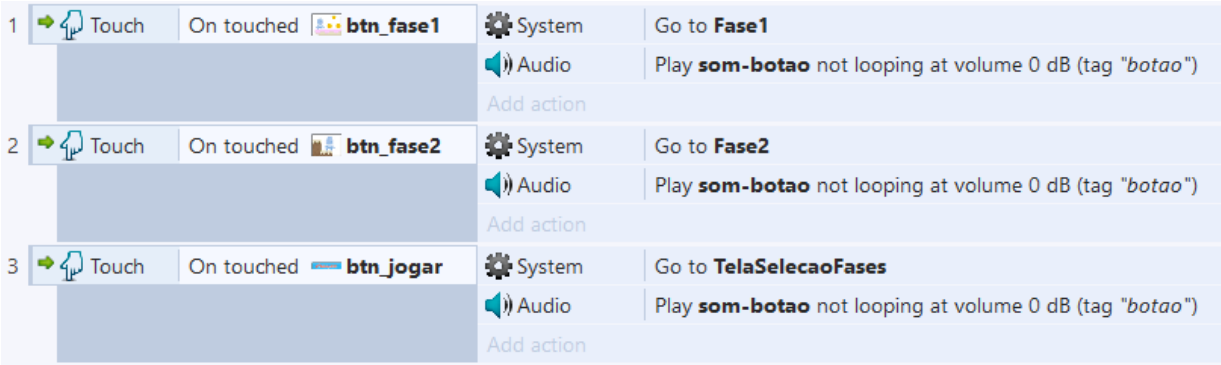

1 Touch On touched btn_fase1 System Go to Fase1
Audio Play som-botao not looping at volume 0 dB (tag "botao")
Add action
2 Touch On touched btn_fase2 System Go to Fase2
Audio Play som-botao not looping at volume 0 dB (tag "botao")
Add action
3 Touch On touched btn_jogar System Go to TelaSelecaoFases
Audio Play som-botao not looping at volume 0 dB (tag "botao")
Add action

Figura 10.7: Eventos dos botões com o efeito sonoro

Agora que aprendemos a inserir som nos eventos, coloque o `som-estrela` com a tag `estrela` no evento da colisão do jogador com a estrela; o `som-chave` com a tag `chave` no evento de colisão do jogador com a chave; e por fim, o `som-inimigo` com a tag `inimigo` no evento da destruição `inimigo1` pelo jogador. Esses três eventos ficarão como mostra a figura a seguir.

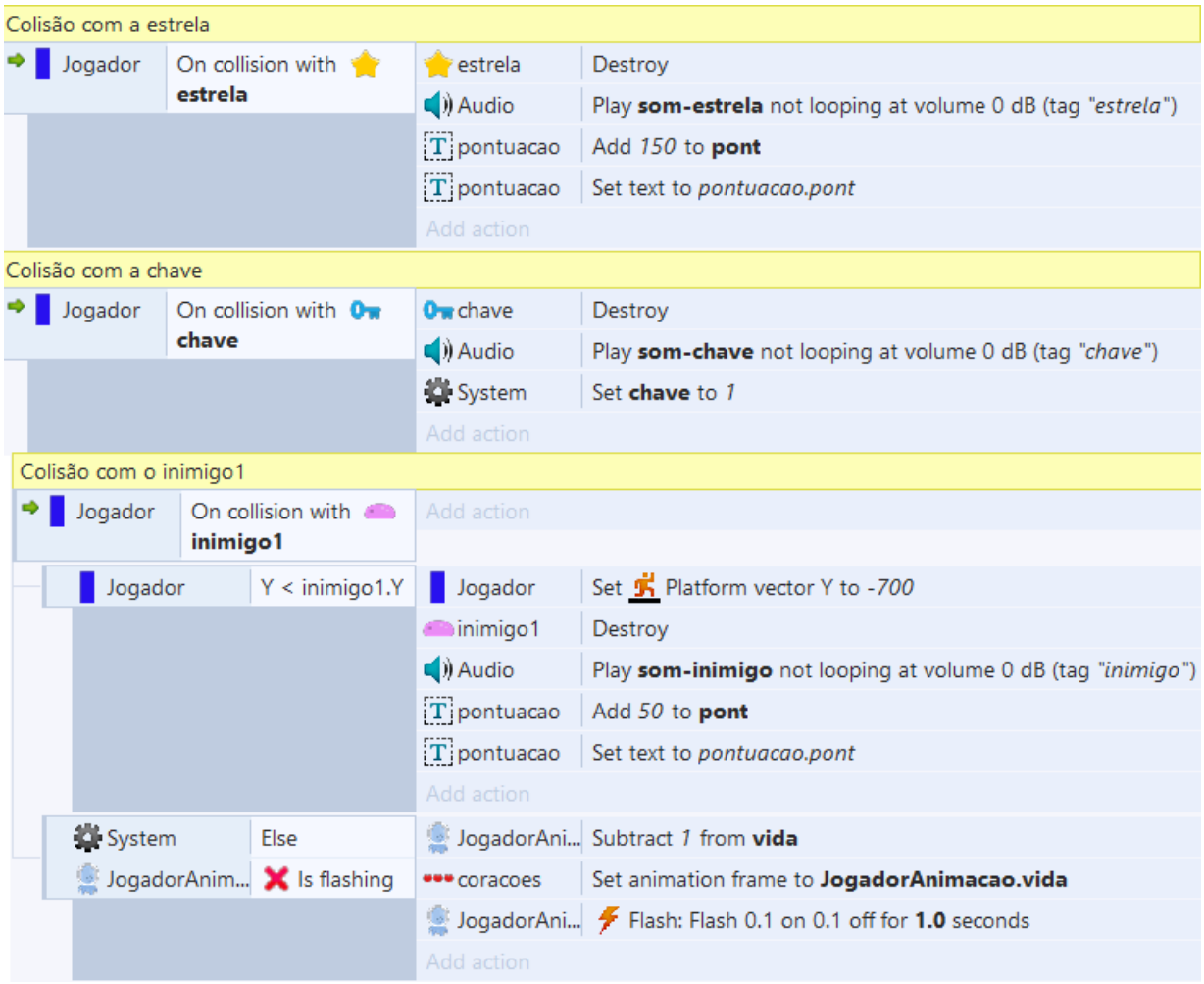


Figura 10.8: Eventos com os outros sons

Pronto, já conseguimos inserir os efeitos sonoros no jogo! Agora precisamos da nossa música de fundo. Veremos como adicioná-la no próximo tópico.

10.3 Adicionando a música de fundo

Vamos inserir agora a música de fundo das nossas fases. Para isso, importe o arquivo de áudio `musica-MattOglseby`, da pasta de recursos `Áudios`, em nossa pasta `Music`. Uma vez importado, queremos que essa música seja executada sempre no começo do layout. Para isso, vamos inserir mais uma ação no evento de inicialização do layout (`On start of layout`), que criamos lá no capítulo *Movimenta ão e anima ão do personagem*.

Escolha o objeto `Audio` e dê a ação `Play` para a música tocar quando o layout iniciar. Nos parâmetros, escolha o arquivo `musica-MattOglseby` no campo `Audio file`, afinal, essa é a música que desejamos tocar nas fases.

Para que a música de fundo nunca pare (nem mesmo quando chegar ao fim), precisamos ligar a sua propriedade `loop`. Então, vá ao campo `loop` e selecione a opção `looping`, assim ela começará a tocar novamente quando terminar. E por fim, coloque um nome para essa música pelo campo `Tag`, como o nome `main`:

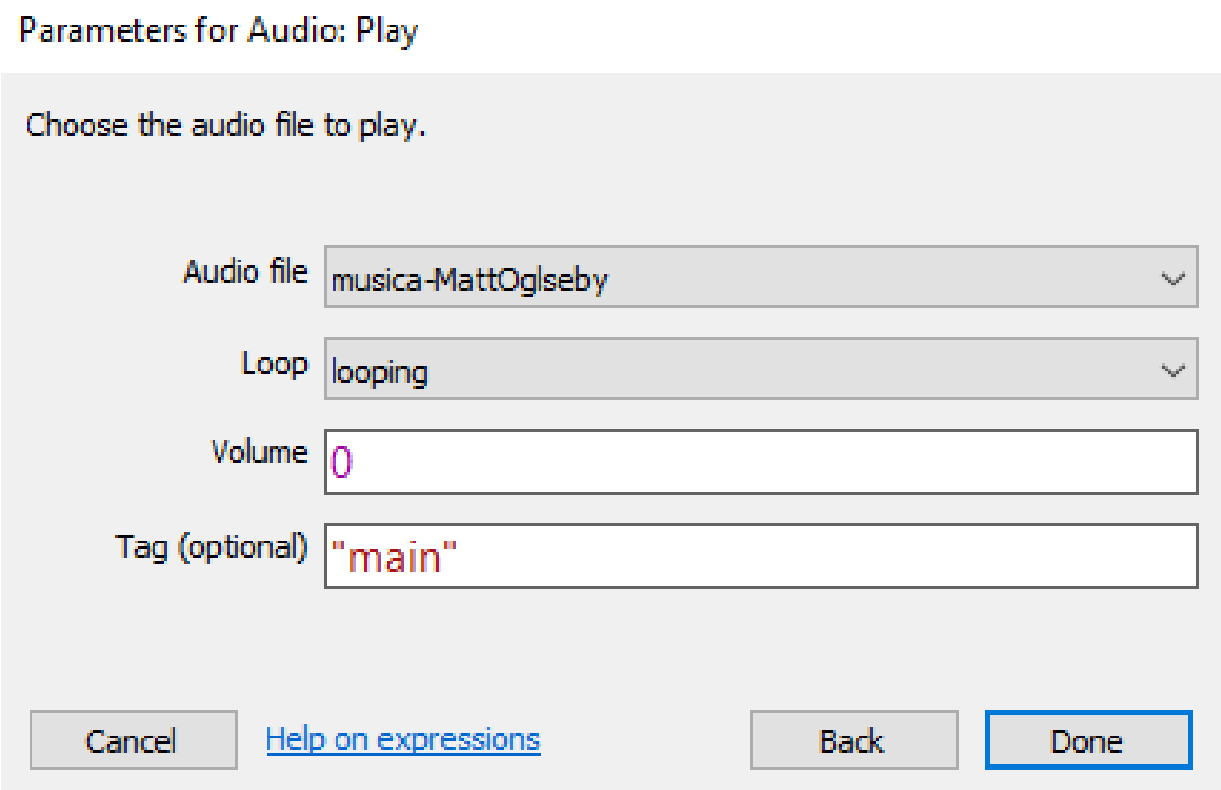


Figura 10.9: Parâmetros da música de fundo

Com isso, cada vez que o layout for iniciado, a música vai tocar e sempre reiniciará quando terminar. Agora temos de colocar um evento para a música parar quando o layout finalizar. Para isso, selecione o objeto de sistema (`System`) e dê a condição `On end of layout` (quando terminar o layout). Para a ação, escolha o objeto `Audio` e dê a ação `Stop all` , como mostra a figura a seguir.

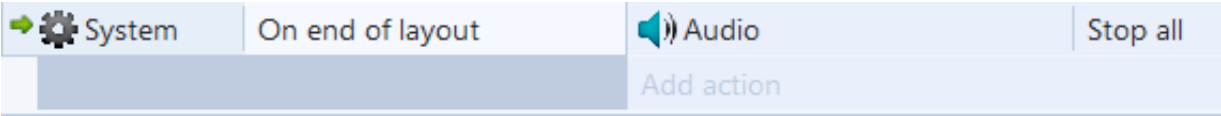


Figura 10.10: Evento On end of layout

Esse evento fará com que todos os arquivos de áudio sejam desligados quando o layout terminar. Queremos que, quando o personagem perder todas as vidas e a tela de *game over* aparecer, a música pare. Para isso, precisamos voltar àquele evento de quando o jogador fica com vida igual a zero, criado no capítulo *Telas*

de pausa e de fim de jogo, e adicionar mais uma ação. Selecione o objeto `Audio` e dê a ação de `Stop`, colocando `main` no campo `Tag`.

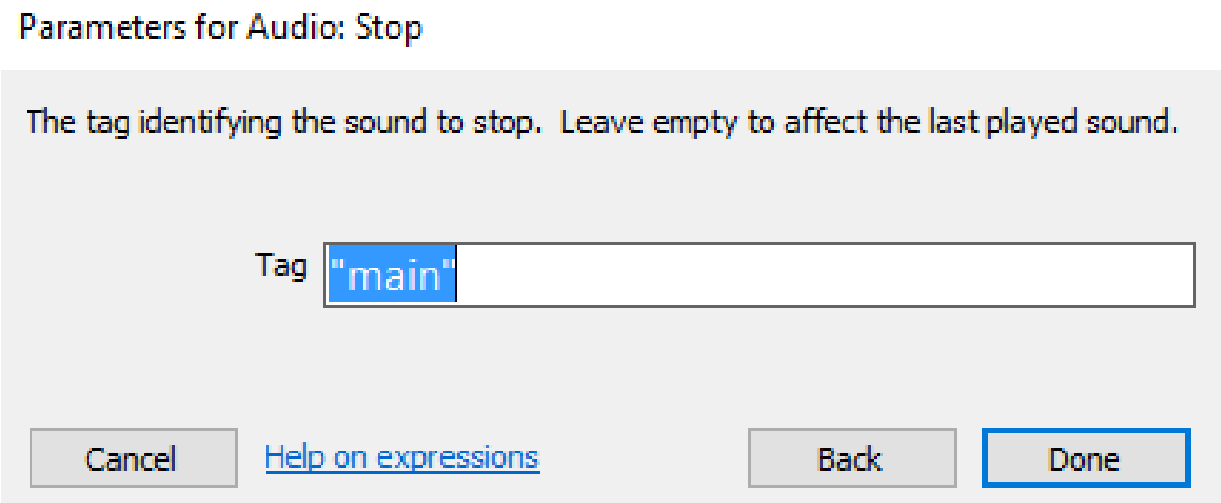


Figura 10.11: Parâmetros do objeto Audio

O evento ficará assim:

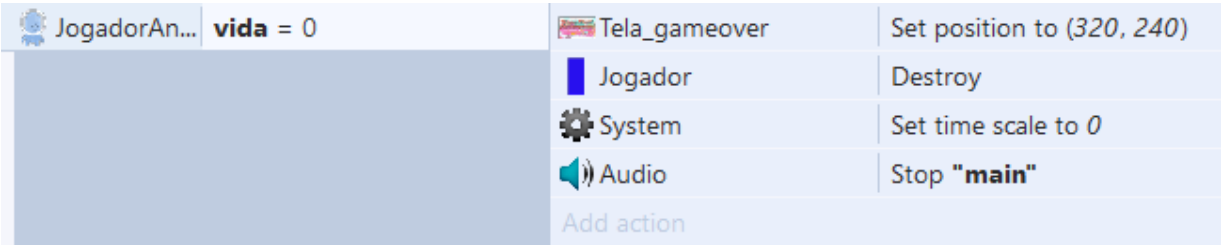


Figura 10.12: Evento de vida igual a zero

Além disso, também queremos que a música pare quando o botão de pausa for clicado e a tela de pausa aparecer. Para realizar isso, vamos adicionar essa mesma ação inserida agora no evento de `Touch` no `btn_pausa`, como mostra a figura a seguir.

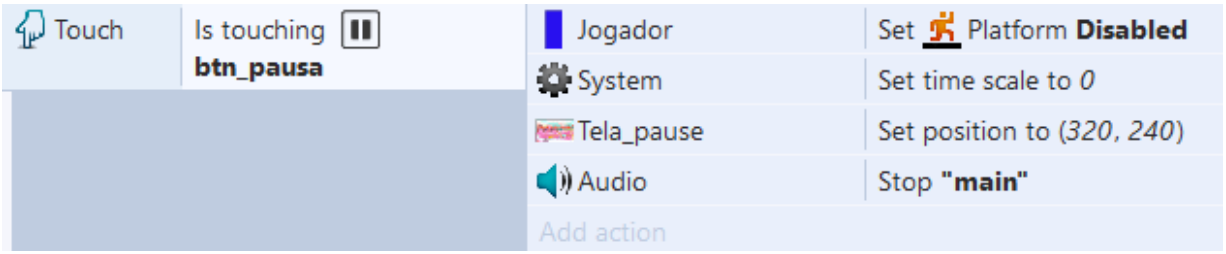


Figura 10.13: Evento do botão de pausa

Agora colocaremos mais uma ação no evento `btn_telaPauseContinue`: quando o botão de continuar da tela de pausa for clicado, desejamos que a música de fundo volte a tocar. Logo, vamos adicionar a mesma ação de tocar a música inserida no evento `On start of layout`, no começo deste capítulo. O evento do botão de continuar ficará assim:

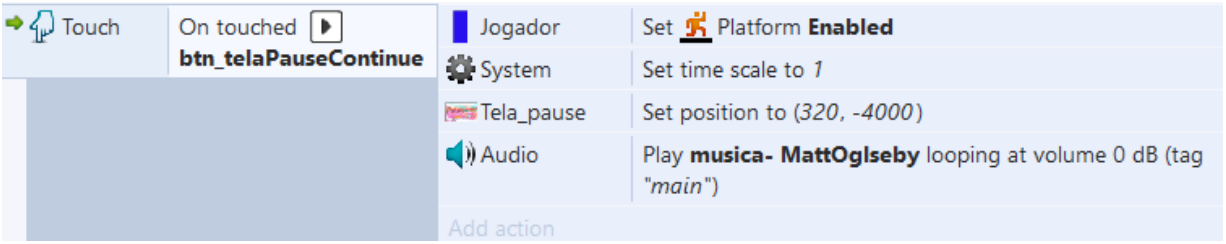


Figura 10.14: Inclusão do botão na tela de pausa

Também temos uma outra situação que precisa de atenção. Pode ser que, quando o jogador for jogar, ele não possa ou queira ouvir a música de fundo e os áudios no momento, por estar no ônibus ou tarde da noite em casa, por exemplo. Por isso, para finalizar, vamos colocar a possibilidade de silenciar os áudios do nosso jogo por meio de um novo botão.

A ideia é que, quando esse botão for clicado uma vez, os áudios sejam mutados. Quando pressionado novamente, os áudios sejam ativados. Portanto, será um botão com dois frames, um para cada uma dessas situações. Vamos lá.

Insira um objeto `Sprite` na layer `UI` com o nome de `btn_som`. Então, no primeiro frame, adicione a imagem `btn-som.png` e, no segundo, a

`btn-no-som.png`, ambas da nossa pasta de recursos `Botões`, como vemos a seguir:

Figura 10.15: Sequência de frames do botão de pausar música

Não esqueça de mudar o `Speed` para `0`, como aprendemos anteriormente, pois só queremos ativar a animação por meio de eventos e não que ela mude sozinha quando iniciar o jogo. Posicione esta `Sprite` que acabamos de inserir ao lado do botão de pausa do jogo:

Figura 10.16: Botão de pausar a música inserido no layout

Agora criaremos o evento para, quando esse botão for pressionado uma vez, todos os áudios do jogo serem silenciados; quando clicado novamente, voltem ao normal. O processo para isso acontecer vai ser muito semelhante ao que já fizemos com os outros botões no decorrer da construção do jogo. Utilizaremos o objeto `Touch` e seus eventos.

Assim, vá à folha de eventos, escolha o objeto `Touch` e dê a condição de `On touched Object` (no objeto clicado), escolhendo o botão `btn_som`. Clique sobre esse evento com o botão direito do mouse e adicione um subevento. Nele, vamos tratar a aparência do botão. Quando ele estiver ativo, terá um quadro de animação

diferente de quando estiver inativo, para que o nosso jogador saiba visualmente quando ele está selecionado ou não.

Vá ao subevento, escolha novamente o botão `btn_som` e dê a condição `Compare frame`. Como por padrão o som estará ligado, precisamos deixar o frame de ligado no começo. Note na animação que ele é o primeiro frame. Coloque os parâmetros para quando o frame for igual a zero, já que é o frame do botão que representa os sons ativos.

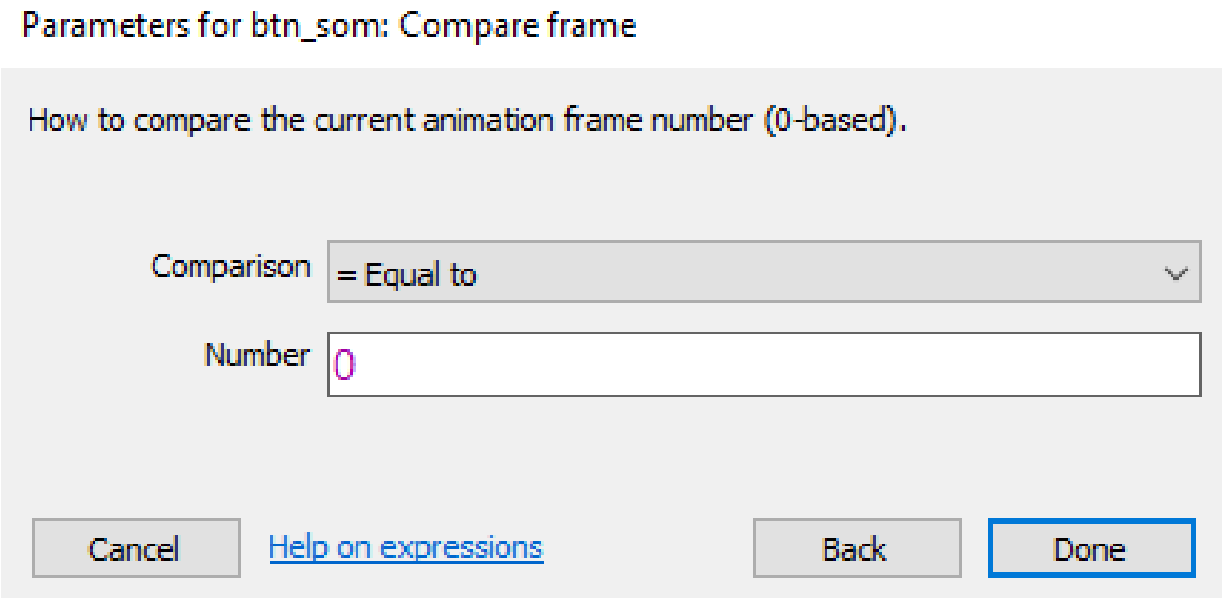


Figura 10.17: Comparando frames

Para a ação, escolha novamente o `btn_som` e dê a ação de `Set frame`, escolhendo o frame 1, que representa o som desligado. Agora, para silenciar o jogo, usaremos uma função apropriada, a `Set silent` do objeto `Audio`. Como o nome já diz, ele torna o jogo mudo, desligando tanto os efeitos sonoros quanto a música de fundo.

Então, escolha o objeto de `Audio` e a ação `Set silent`, como descrevemos. No campo `Mode`, coloque a opção `Silent`, que diz para que todos os sons e músicas fiquem mudos, como mostra a seguir:

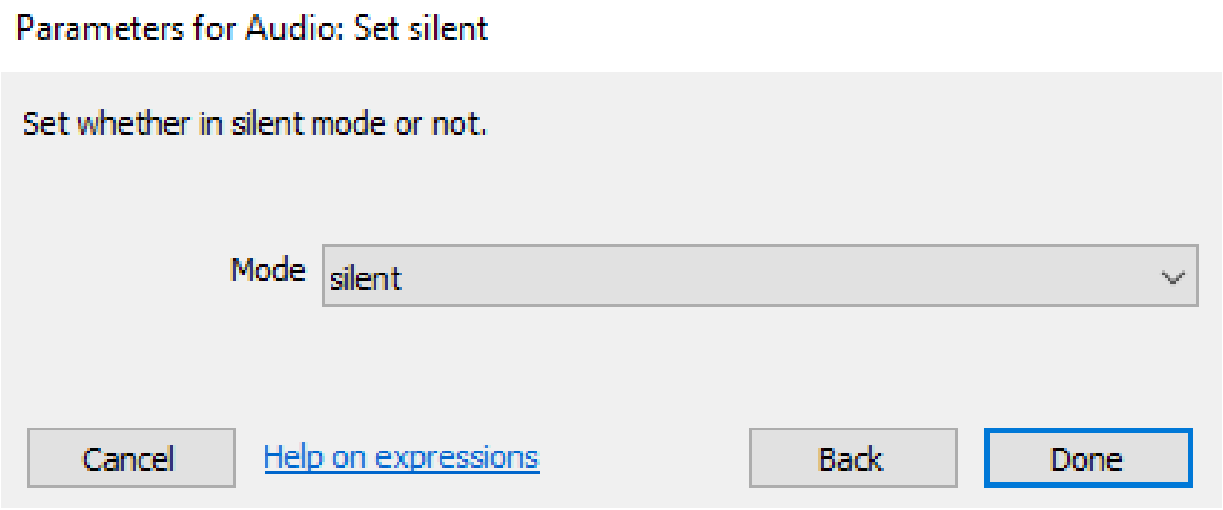


Figura 10.18: Set silent

Quando o botão `btn_som` for clicado e os áudios estiverem ativados (`frame 0`), setamos o frame para `1` e eles são silenciados. Queremos que, quando ele for pressionado novamente, o frame seja setado para `0` e os áudios voltem a tocar. Para isso, clique novamente sobre o evento principal `Touch` , com o botão direito do mouse, e insira mais um subevento. Faremos basicamente o inverso do que fizemos no primeiro passo.

Escolha o objeto de sistema `System` e a condição `Else` . Na ação, insira novamente o `btn_som` e, mais uma vez, dê a ação de `Set frame` , colocando desta vez o frame `0` . Na outra ação, insira o objeto de `Audio` , dê a ação `Set silent` e, no campo `Mode` , coloque a opção `No silent` , para despausar todos os áudios do jogo.

Pronto, o evento de pausar e despausar todas as melodias ficará como a figura a seguir:

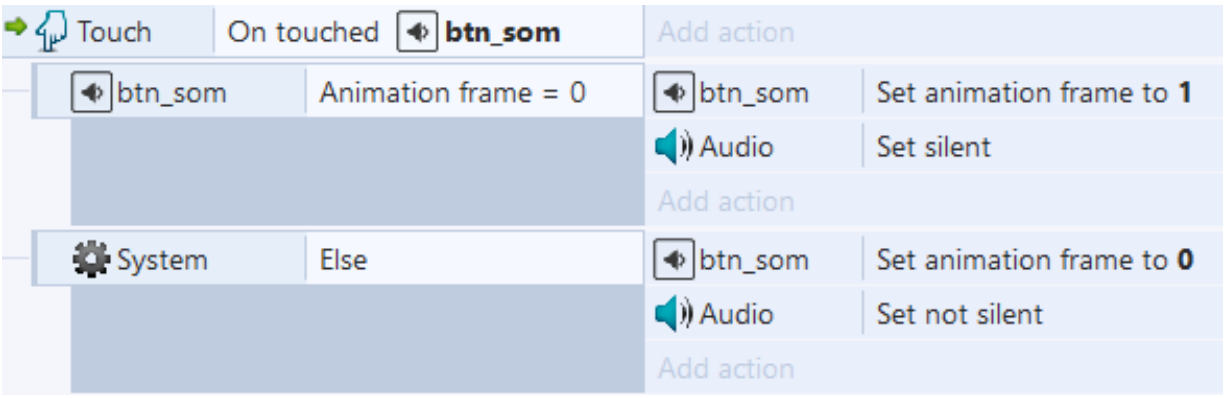


Figura 10.19: Evento de pausar e despausar a música de fundo

Colocamos acima o botão responsável por silenciar e ativar os áudios do jogo, porém, no Construct 2, não temos a possibilidade de aumentar/ diminuir o som. Para isso, o jogador terá de elevar ou baixar o som do próprio aparelho/ dispositivo no qual estiver jogando.

10.4 Tela inicial e a seleção de fases

Agora que já configuramos todo o mecanismo de volume das melodias do jogo, vamos finalizar o capítulo inserindo as músicas das telas de início e de seleção de fases. O processo é muito semelhante ao que já fizemos até agora. A diferença aqui é que não poderemos criar tudo na folha de eventos que estivemos trabalhando no decorrer do jogo. Para estas telas, colocaremos os eventos em suas folhas de eventos específicas.

Começando pela tela inicial, acesse a folha de eventos `Telas` e adicione o evento `On start of layout` do objeto `System`. Feito isso, adicione a ação `Play` do objeto `Audio` e selecione a música `musica_telaInicial` com a `tag` de mesmo nome. Também não se esqueça de ativar a opção `looping` para que ela toque indefinidamente.

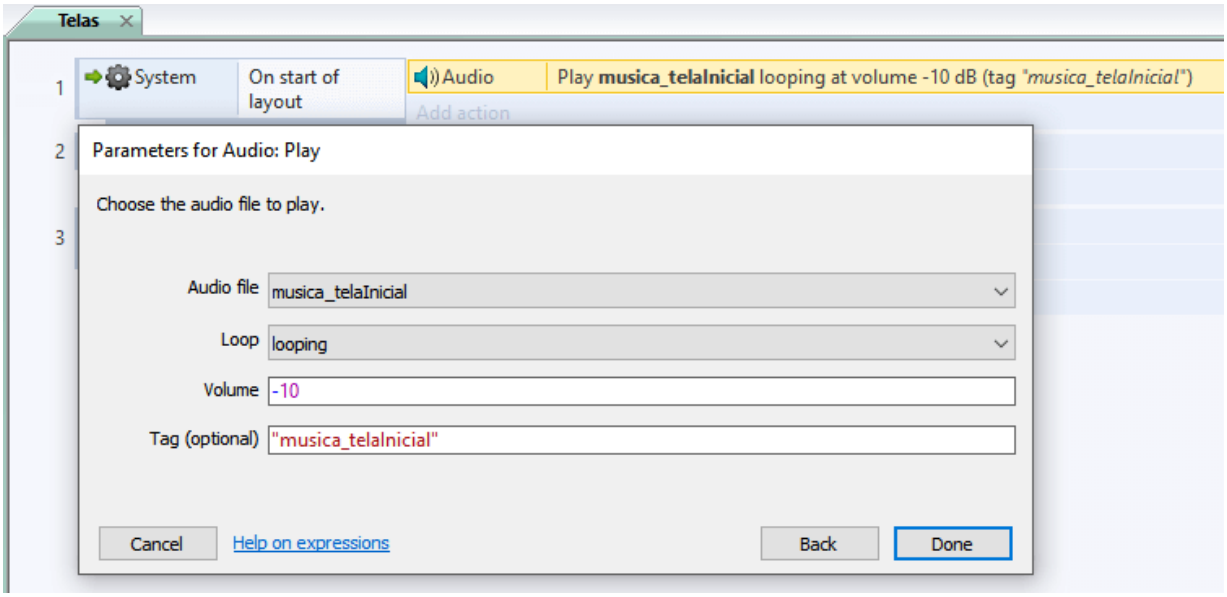


Figura 10.20: Adicionando uma música de fundo à tela inicial

Feito isso, adicione o evento `On end of layout`, também do objeto `System`. Desta vez, adicione a ação `Stop` do objeto `Audio` e coloque a `tag musica_telaInicial`. Desta maneira, conseguiremos evitar que as músicas do jogo toquem uma em cima da outra. Ao final, tudo deverá ficar assim:

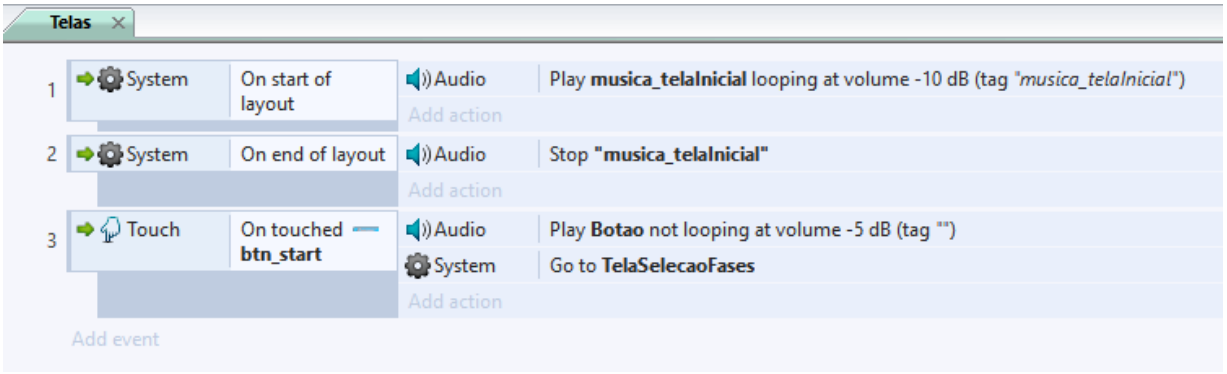


Figura 10.21: Evitando que as músicas entrem em conflito

Muito bem. Agora basta repetir o mesmo processo para a tela de seleção de fases. Escolha a folha de eventos `TelaSelecaoFases` e adicione os eventos do objeto `System` para início e fim do layout. Para a música, escolha a opção que deixamos na pasta, a `musica_telaSelecaoDeFases`. Ao final, deverá ficar assim:

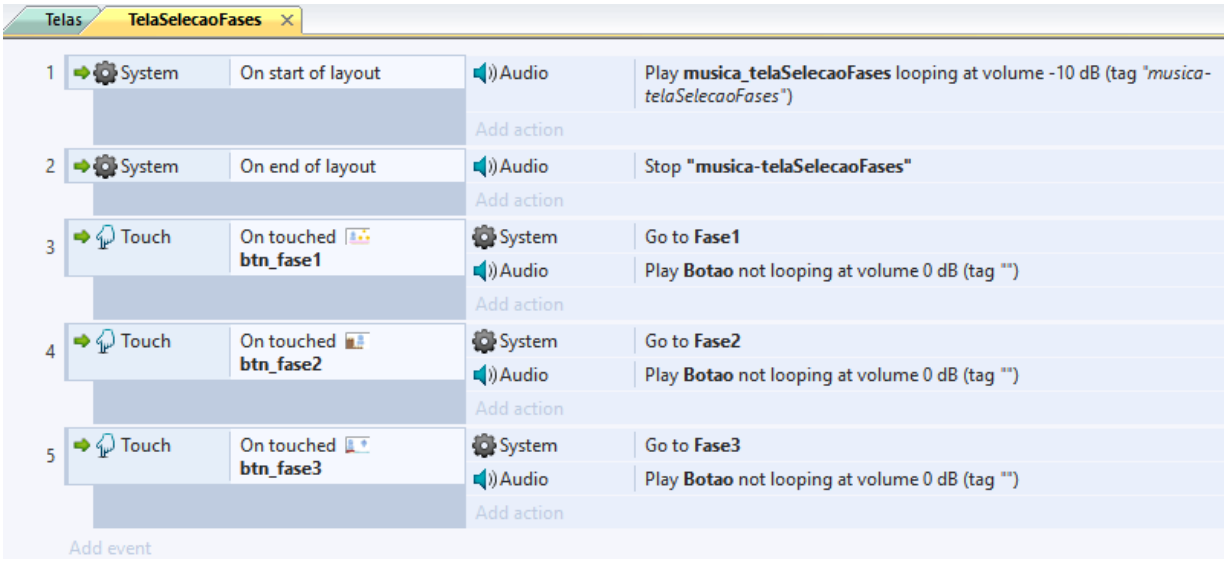


Figura 10.22: Adicionando uma música de fundo à tela de seleção de fases

10.5 Fim de jogo

Ufa! E assim finalizamos o processo de criação do nosso jogo! Esperamos que agora você já tenha aprendido a como desenvolver tudo o que um jogo de plataforma precisa, desde criar fases e mecanismos de movimentação do personagem, dos inimigos e da plataforma; inserir funcionalidades nos botões; importar sons e música; entre muitas outras coisas que vimos no decorrer dessa jornada.

No próximo capítulo, vamos aprender a exportar o nosso jogo e colocá-lo na internet para mostrar nossa obra ao mundo!

Capítulo 11
Exportação e distribuição

Agora que finalizamos todo o desenvolvimento do nosso jogo, chegamos à última etapa: a exportação. Este processo compacta todos os nossos arquivos de música, sons, imagens, layouts e códigos e converte-os em um formato específico (dependendo da plataforma escolhida). Como citamos lá no início, o Construct 2 nos oferece diversas opções de exportação, tais como:

- Web (HTML5);
- Wii U;
- Mobile (iOS, Android, Windows Phone, Blackberry);
- Desktop (Windows, Mac OSX, Linux);
- Extensões e aplicativos para navegadores (Firefox, Chrome) ;
- Facebook;
- Amazon Appstore.

Infelizmente, na versão gratuita, há poucas opções de exportação. Temos somente disponíveis as extrações para Web, Facebook e Windows Store, além da própria exportação para a Scirra Store, o *marketplace* de jogos da desenvolvedora do Construct 2.

O modo de exportação mais simples e eficaz é o Web (HTML5). O Facebook exige uma série de configurações extras (https://www.scirra.com/tutorials/58/how-to-make-a-facebook-game), e a Chrome Web Store exige uma licença de desenvolvedor (https://www.scirra.com/tutorials/68/publishing-to-the-chrome-web-store). Para a exportação com que trabalharemos aqui, conseguiremos fazer tudo gratuitamente.

Neste capítulo, veremos como utilizar a exportação para Web (HTML5) e como usar o código gerado para distribuir o jogo na internet de forma gratuita, utilizando o BitBalloon (https://www.bitballoon.com/), serviço de hospedagem gratuita.

11.1 Exportando o jogo

O processo de exportação é muito mais simples do que parece. Para exportar o projeto, basta clicar na opção `Export Project` (exportar projeto), localizada em `File` no menu superior. Feito isso, a seguinte janela será exibida:

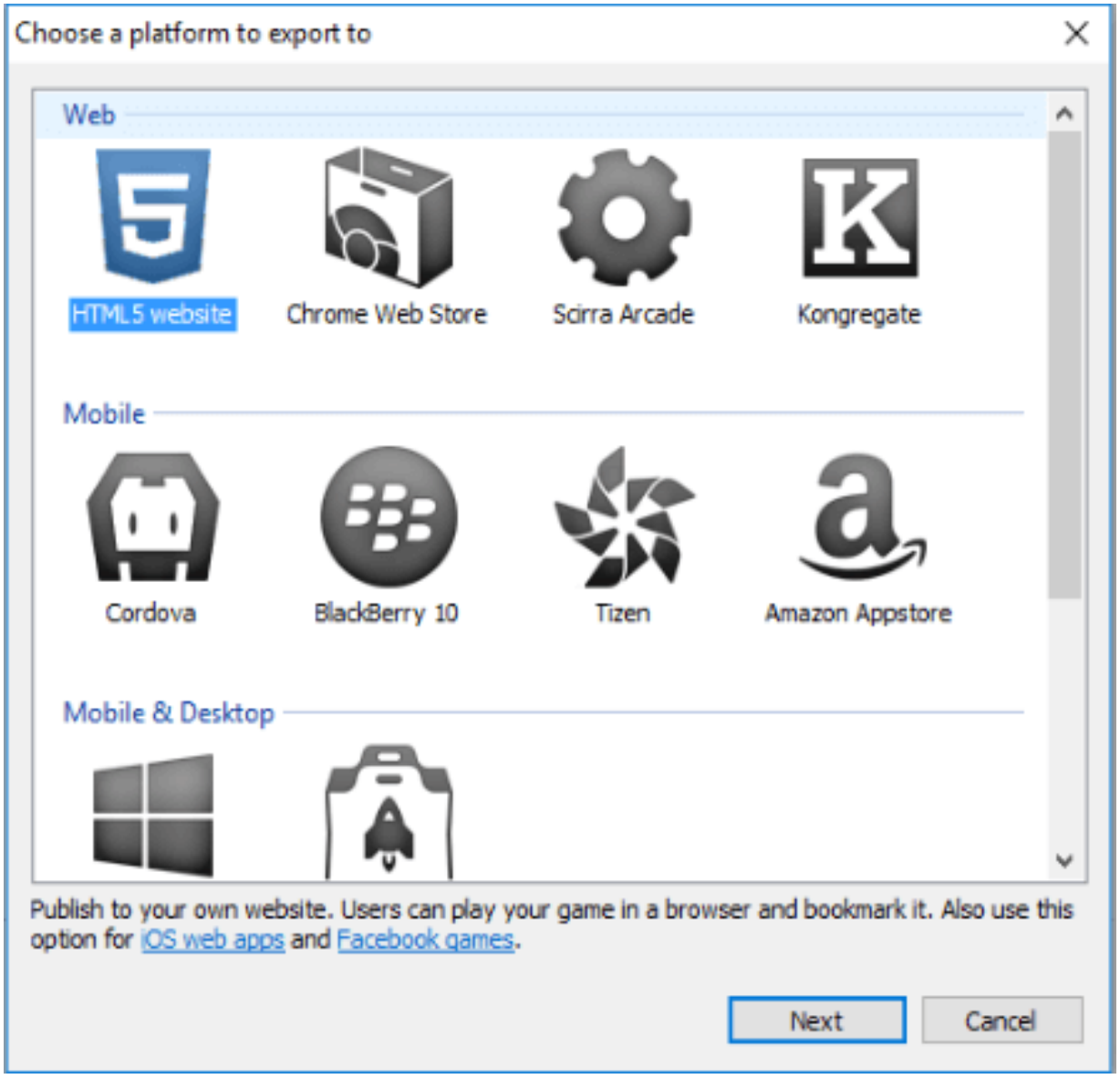


Figura 11.1: Exportando para HTML5

Note que na imagem há várias opções, mas na versão gratuita somente a opção `HTML 5 Website` estará disponível. Selecione-a e

clique em `Next`. Na próxima janela, escolha aonde deseja exportar os seus arquivos no campo `Export files to`. Por padrão, ele deixará preenchido com o local onde o seu jogo está sendo desenvolvido. Recomendamos exportar em alguma outra pasta para não causar confusões.

Nas opções `Subfolder for images` (subdiretório para imagens) e `Subfolder for project files` (subdiretório para arquivos do projeto), deixe com a opção já preenchida por padrão – `/images` e `/media`, respectivamente. A título de curiosidade, essas opções servem para que você altere o local onde os arquivos do seu projeto serão salvos dentro da estrutura do projeto. Dificilmente você terá de alterar esse valor.

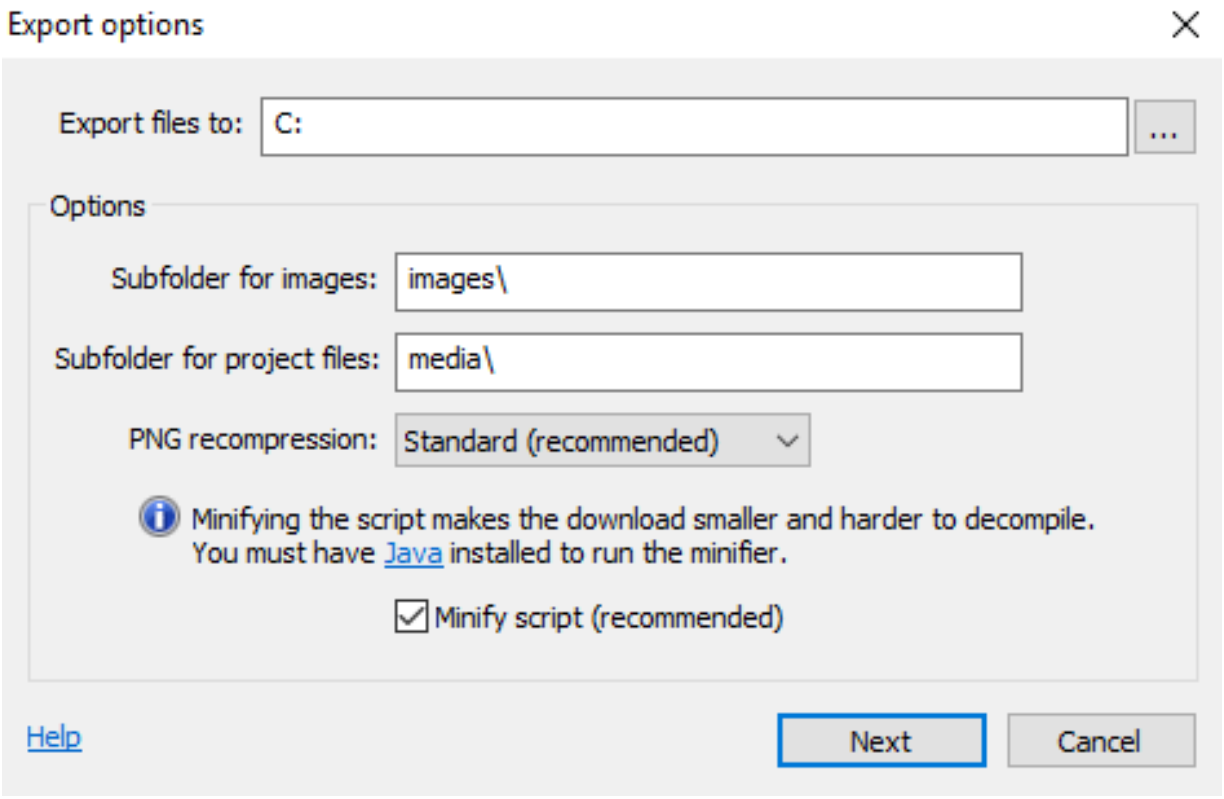


Figura 11.2: Opção de pasta

No campo `PNG recompression` (compressão PNG), podemos escolher como as imagens contidas no nosso jogo serão comprimidas (diminuídas). Existem três opções disponíveis:

- `None` : nenhuma compressão será aplicada, e as imagens ficarão do mesmo tamanho;
- `Standard` : as imagens serão comprimidas no tamanho recomendado;
- `Brute` : as imagens serão comprimidas ao máximo, mas isso pode causar uma demora na exportação do jogo (e elas podem perder parte da qualidade).

A compressão de imagens é importante para deixarmos o nosso jogo otimizado. Quanto mais leves forem as imagens, mais rápida elas são carregadas. Como nossas figuras não são pesadas, deixaremos selecionada a opção `Standard` .

A última opção disponível é o `Minify script` (minificar script). A prática de minificar o código-fonte é comum no desenvolvimento de software, e tem como objetivo tornar o código menor e ofuscado. Como resultado, o jogo é carregado mais rápido, e o código torna-se extremamente confuso para qualquer um que tente decodificá-lo. Lembre-se de que, no final das contas, todo o código gerado pelo Construct 2 é JavaScript. Vamos deixar esta opção marcada.

> **REQUISITOS PARA MINIFICAR O C DIGO**
>
> Para que o código do jogo seja minificado, é necessário ter o Java instalado na máquina. Caso não o tenha, clique em https://www.java.com/pt_BR/ e baixe o instalador. Feito isso, basta executá-lo para toda a instalação ser feita.

Depois, clique novamente em `Next` . Na próxima janela, escolheremos o tamanho do template que desejamos que o jogo seja visualizado. Esta opção é muito útil para caso você queira incorporar o código do jogo em alguma outra página da internet. Para o nosso caso, queremos que ele ocupe toda a tela disponível. Para isso, selecione a opção `Normal Style` .

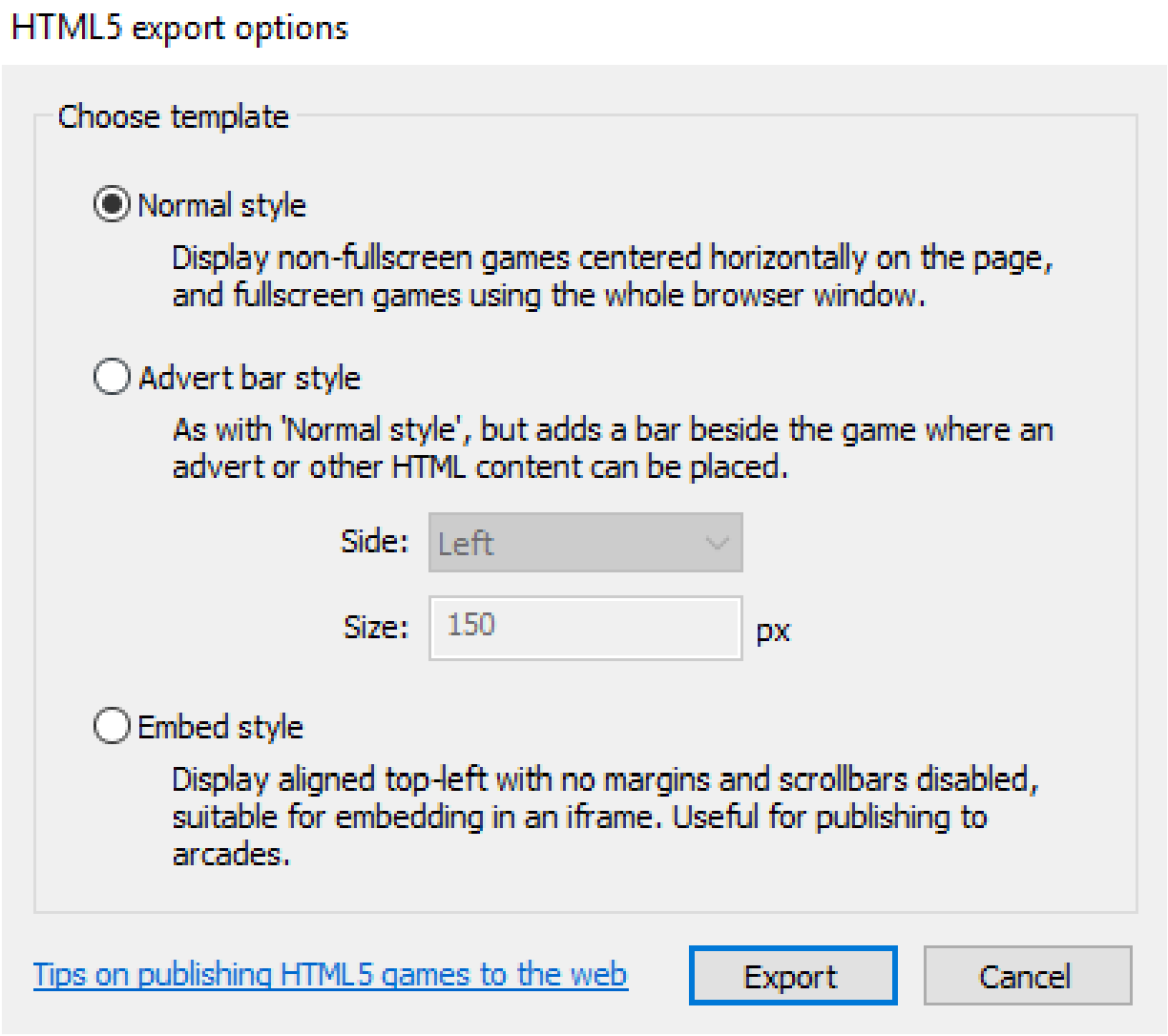

HTML5 export options
Choose template
Normal style
Display non-fullscreen games centered horizontally on the page, and fullscreen games using the whole browser window.
Advert bar style
As with 'Normal style', but adds a bar beside the game where an advert or other HTML content can be placed.
Side:
Left
Size:
150
px
Embed style
Display aligned top-left with no margins and scrollbars disabled, suitable for embedding in an iframe. Useful for publishing to arcades.
Tips on publishing HTML5 games to the web
Export
Cancel

Figura 11.3: Opção de pasta

Feito todos os ajustes, clique na opção `Export` . A partir daí, a plataforma fará toda a conversão do nosso jogo. Esse processo não deverá levar mais do que alguns segundos (quanto maior o jogo, maior o tempo para conseguir exportá-lo). Quando terminar, a janela seguinte será exibida mostrando que tudo ocorreu como esperado:

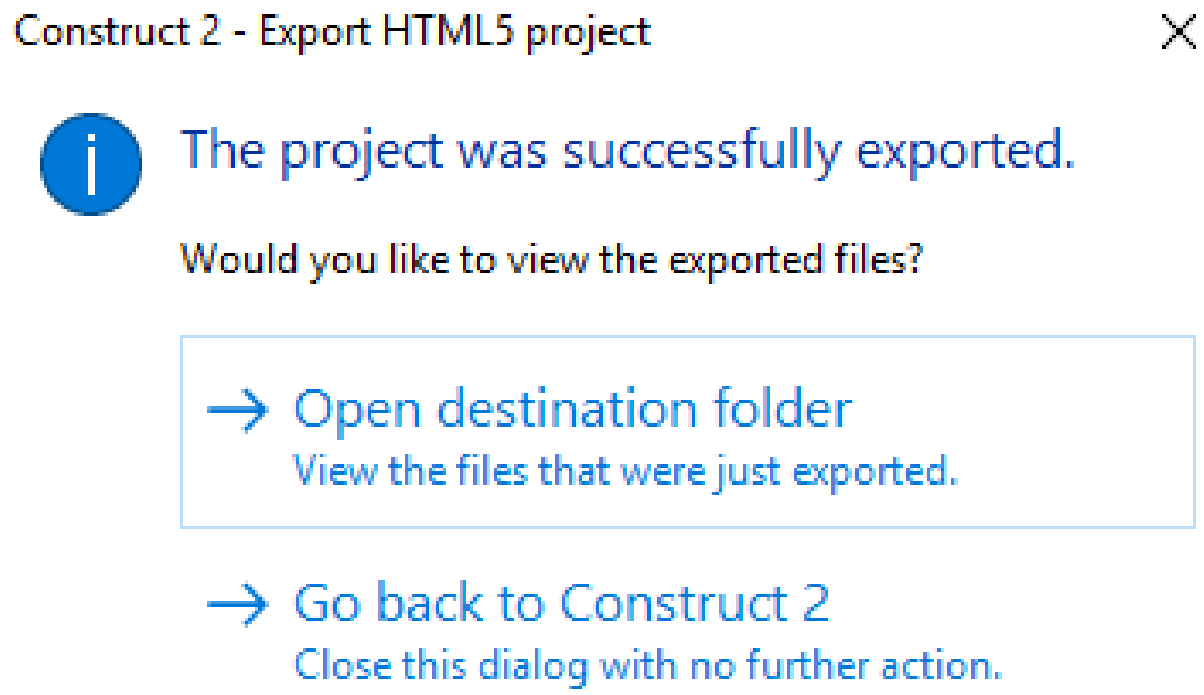

Construct 2 - Export HTML5 project
The project was successfully exported.
Would you like to view the exported files?
Open destination folder
View the files that were just exported.
Go back to Construct 2
Close this dialog with no further action.

Figura 11.4: Exportação com sucesso

Pronto, o nosso jogo foi exportado! Escolha a opção `Open destination folder` e veja a estrutura criada. Se você já tiver uma experiência com desenvolvimento de sites, notará que o código gerado é basicamente um site. Clique no arquivo `index.html`, e perceba que o jogo será aberto no navegador e executado.

Então, com o jogo criado e exportado, chegou a hora de expô-lo na internet para que seus amigos e familiares possam jogar.

11.2 Hospedando o jogo na internet

Agora que exportamos o jogo, queremos distribuí-lo para que outras pessoas possam jogá-lo. Uma das formas mais fáceis e rápidas para isso é enviá-lo para a plataforma de hospedagem BitBalloon. Ela oferece hospedagem gratuita para sites estáticos. Como o

nosso jogo foi exportado para HTML5, a plataforma cai como uma luva.

Caso você já tenha um serviço de hospedagem de sites (como UOL Host, GoDaddy ou HostGator), é possível utilizá-los também sem problemas. Mas para quem nunca lidou com hospedagem de websites na internet, recomendamos seguir os próximos passos.

Primeiramente, entre no site https://www.bitballoon.com/, e crie uma conta gratuita clicando na opção `Sign Up for Free`. Após isso, você será levado para a página a seguir, na qual poderá enviar os seus projetos em HTML5.

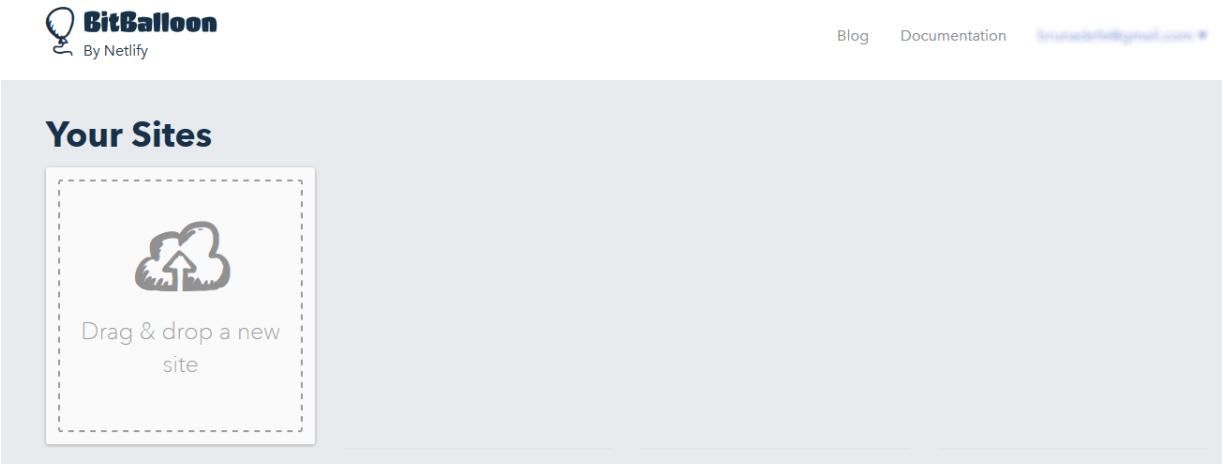


Figura 11.5: BitBalloon

É muito simples enviar o projeto para a plataforma. Compacte o código gerado dentro de um `zip`, então arraste-o para a área que diz: `Drag e drop a new site` (arraste e solte um novo site). Pronto, depois disso, é só esperar alguns segundos o *uploading*.

> O BitBalloon aceita somente um arquivo por vez no upload, por isso ele precisa estar em um `zip`. Além disso, arquivos `.war` ficam compactados em outro formato, e o BitBalloon não o entende.

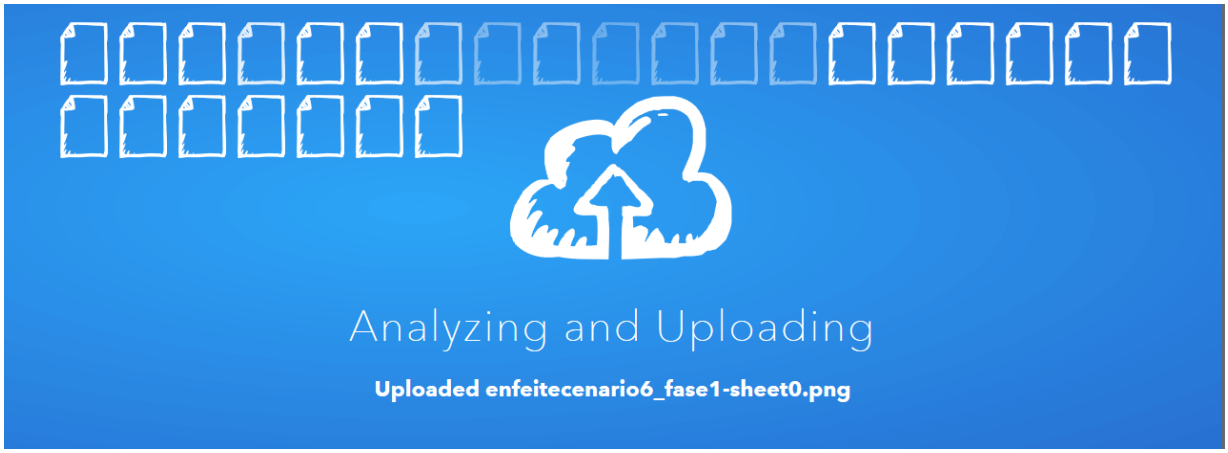


Figura 11.6: Uploading

Após concluir o upload, será gerado um link para o projeto com um nome aleatório:

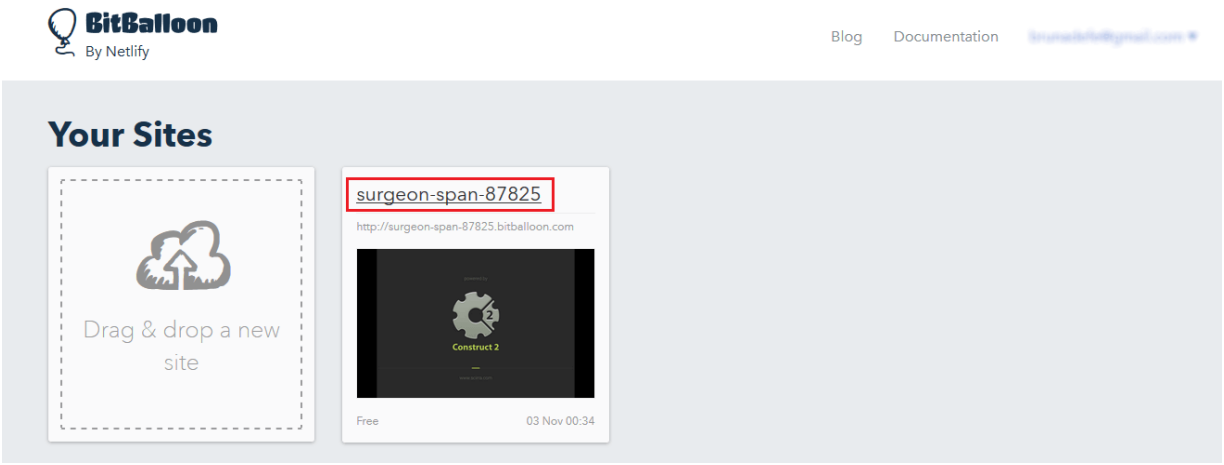


Figura 11.7: Upload concluído

Para modificá-lo, basta clicar no nome aleatório criado e, na próxima página, editar o nome no campo `Edit name` (editar nome).

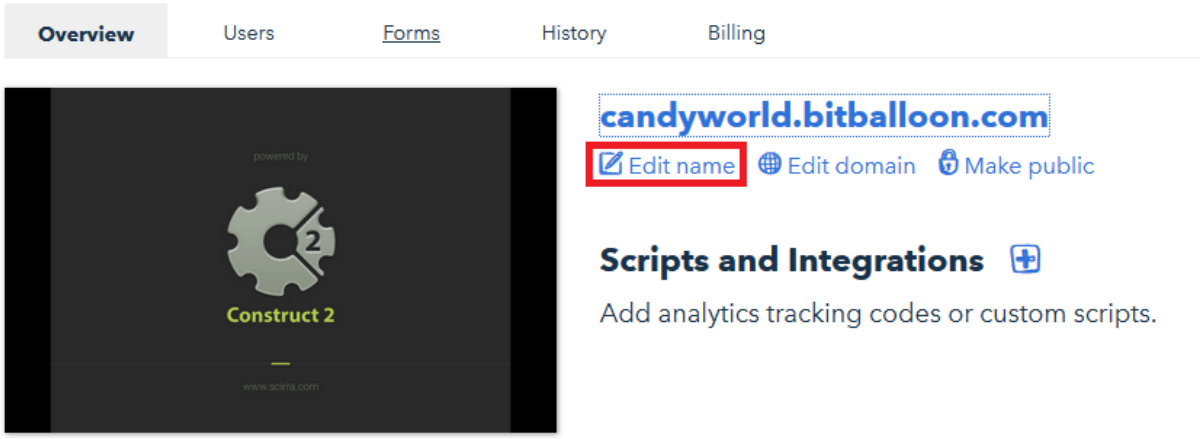


Figura 11.8: Editando o nome

Pronto, um link para o jogo foi criado com o nome escolhido, e agora você já pode jogar ou enviar para quem quiser! Caso você queira subir uma nova versão do mesmo jogo, basta repetir o processo, assim como para subir novos jogos.

A seguir, acesse o link do nosso jogo hospedado no BitBalloon: http://candyworld.bitballoon.com/.

Capítulo 12
Dicas finais

Neste capítulo, daremos algumas dicas que poderão ser úteis no desenvolvimento dos seus futuros jogos.

12.1 Onde conseguir recursos

Grandes produtoras de jogos (como Electronic Arts, Sony, Ubisoft e tantos outros estúdios) possuem grandes equipes responsáveis por lidar com cada um dos aspectos de um jogo. Uma equipe cuida da animação, outra da música, outra da programação, e assim por diante. Quando estamos trabalhando de forma independente, precisamos tratar de todos estes aspectos sozinhos. Para facilitar esse trabalho, podemos contar com recursos distribuídos na internet.

Podemos encontrar de tudo na internet, desde músicas de fundo, sons de efeitos, desenhos de personagens, códigos de template etc. Pacotes de recursos como o que utilizamos neste livro também podem ser obtidos, mas é preciso muito cuidado para não violar os direitos autorais de criação do dono desses recursos.

Muitos não levam isso a sério e usam qualquer um que acha na internet, mas lembre-se de que isso é contra a lei. Entretanto, isso não significa que você precisará gastar fortunas para conseguir acessar esses materiais.

Há uma série de sites nacionais que oferecem, por um baixo custo (ou de graça), recursos para serem utilizados. Uma vez comprados, podem ser usados sem medo para construção de jogos, mesmo que sejam de caráter comercial.

A seguir, veja uma lista com alguns sites nacionais que podem ser usados para obter recursos para os seus jogos:

- **Time to Play** (http://timetoplay.com.br/) – Empresa focada na criação de jogos com códigos-fontes comentados, templates e recursos para jogos.
- **Polar Games** (http://polargames.com.br/) – Empresa focada na criação de cursos de Construct 2.
- **Parametade Games** (http://www.parametade.com/) – Projeto de introdução a desenvolvimento de games por meio de videoaulas e códigos-fontes.

Existem também alguns sites internacionais, como:

- **Incompetech** (https://incompetech.com/) – Site com várias músicas gratuitas e com licença livre para uso nos jogos.
- **Free Play Music** (http://freeplaymusic.com/) – Site que também oferece várias músicas gratuitas com licença livre.
- **BFXR** (https://www.bfxr.net/) – Programa gratuito para fazer efeitos sonoros para jogos.
- **Orange Free Sounds** (http://www.orangefreesounds.com/) – Site que oferece músicas e sons gratuitos com licença livre.

12.2 Conversão de áudio

Como explicamos no capítulo *Música e sons*, para facilitar a importação de áudios no Construct 2, é recomendado que ele esteja no formato `.wav`. Uma forma fácil, rápida e gratuita para converter áudios de qualquer formato para qualquer formato é o Cloud Convert (https://cloudconvert.com/).

Logo que entramos no site, veremos o botão `Select Files`, como mostra a figura a seguir:

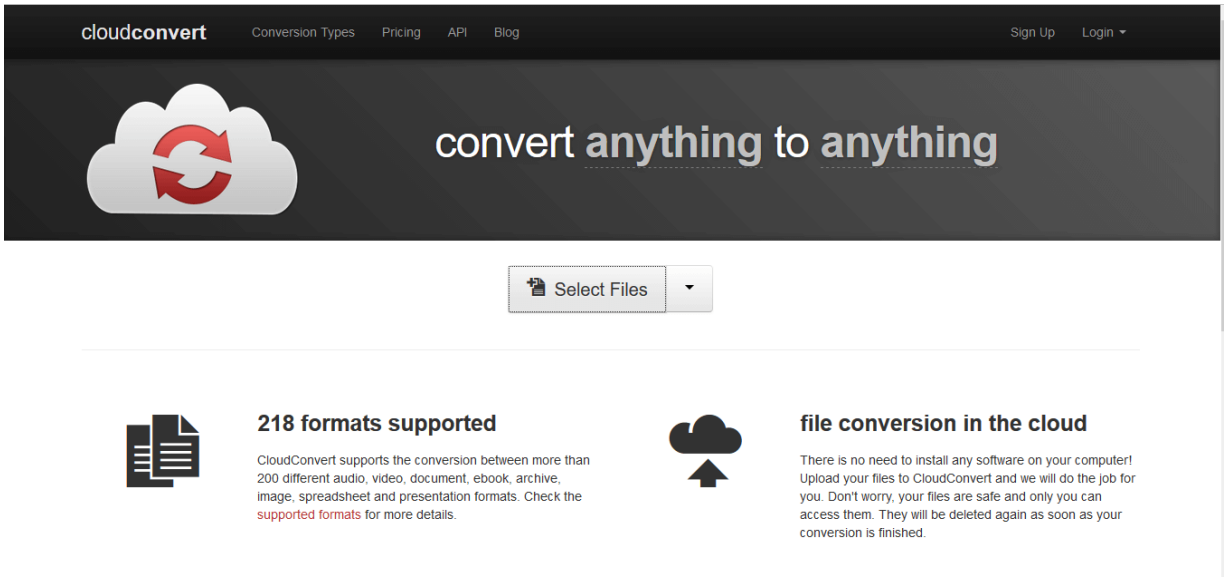

cloudconvert
Conversion Types
Pricing
API
Blog
Sign Up
Login
convert anything to anything
Select Files
218 formats supported
CloudConvert supports the conversion between more than 200 different audio, video, document, ebook, archive, image, spreadsheet and presentation formats. Check the supported formats for more details.
file conversion in the cloud
There is no need to install any software on your computer! Upload your files to CloudConvert and we will do the job for you. Don't worry, your files are safe and only you can access them. They will be deleted again as soon as your conversion is finished.

Figura 12.1: Cloud Convert

Ao clicar nele, é aberta uma janela para procurar o arquivo de áudio que você deseja fazer a conversão dentro dos seus diretórios. Escolha-o e clique em abrir. Depois de o sistema verificar qual o formato do áudio importado, você terá a opção de escolher qual é o formato em que quer convertê-lo:

Figura 12.2: Conversão

Em seguida, basta clicar no botão `Start Conversion` para a conversão começar:

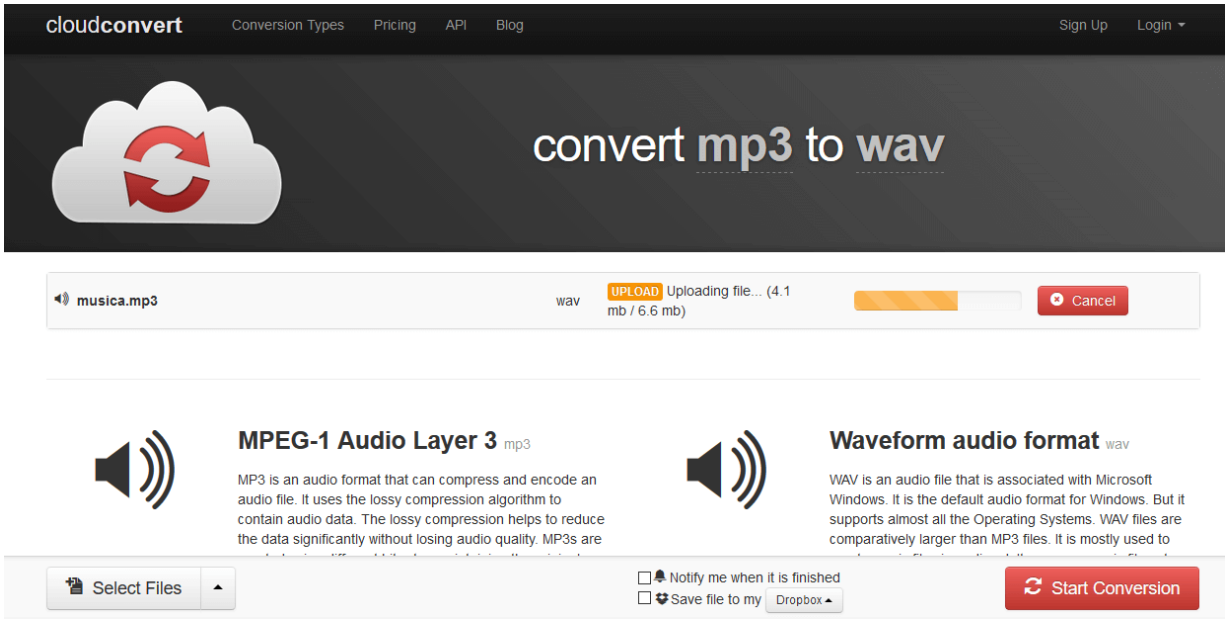

cloudconvert
Conversion Types
Pricing
API
Blog
Sign Up
Login
convert mp3 to wav
musica.mp3
wav
UPLOAD Uploading file... (4.1 mb / 6.6 mb)
Cancel
MPEG-1 Audio Layer 3 mp3
MP3 is an audio format that can compress and encode an audio file. It uses the lossy compression algorithm to contain audio data. The lossy compression helps to reduce the data significantly without losing audio quality. MP3s are
Waveform audio format wav
WAV is an audio file that is associated with Microsoft Windows. It is the default audio format for Windows. But it supports almost all the Operating Systems. WAV files are comparatively larger than MP3 files. It is mostly used to
Select Files
Notify me when it is finished
Save file to my
Dropbox
Start Conversion

Figura 12.3: Conversão iniciada

Após isso, basta fazer o download do arquivo em seu computador e usá-lo em seus projetos!

12.3 Referências

Manual oficial do Construct 2 – https://www.scirra.com/manual/1/construct-2

Tutorial da Scirra - *Beginner's guide to Construct 2* – https://www.scirra.com/tutorials/37/beginners-guide-to-construct-2

Produção de jogos - *Construct 2: o Guia Completo*, por Raphael Dias – http://producaodejogos.com/construct-2/

Meetup Construct 2 - SP – https://www.meetup.com/pt-BR/Construct2-Sao-Paulo/

Grupo do Facebook - *Construct 2 Brasil* –
https://www.facebook.com/groups/Construct2BR/

Capítulo 13
Construct 3

O Construct 3 está na versão beta para o público desde abril de 2017. Ainda não saíram informações de quando essa nova versão se tornará estável, mas já podemos testá-la para verificar as diferenças e as novidades.

Vale ressaltar que o Construct 2 ainda vai ser mantido pela Scirra a longo prazo, e tudo o que foi ensinado no decorrer deste livro continuará válido. Porém, as novidades das funcionalidades só serão implementadas no Construct 3.

Figura 13.1: Visual do Construct 3

13.1 Diferenças

A principal mudança, e que foi bastante criticada pelo público, é o preço. Agora, é preciso pagar por ano para ter acesso às versões pagas do Construct, diferente do Construct 2, já que nele era necessário pagar somente uma vez para adquiri-lo. Os planos serão:

- **Personal:** para uso individual e sem fins lucrativos = R$ 343,39 por ano.
- **Business:** para empresas e organizações de todos os tamanhos e tipos, com fins lucrativos e sem restrição de valores = R$ 499,89 por ano.
- **Education:** somente para o uso de estudantes e instituições de ensino = R$ 47,79 por mês ou R$ 139,09 por ano.

Temos também a opção de usar a versão gratuita do Construct 3. Mas ela é bem mais limitada do que versão gratuita do Construct 2. Das principais diferenças destacam-se as limitações:

- No Construct **2** gratuito, podemos ter até 100 eventos, 4 layers e 2 efeitos.
- No Construct **3** gratuito, podemos ter somente até 25 eventos, 2 layers e 2 efeitos.

Outra diferença bem relevante também é que agora o Construct 3 roda dentro do navegador, como vemos a seguir:

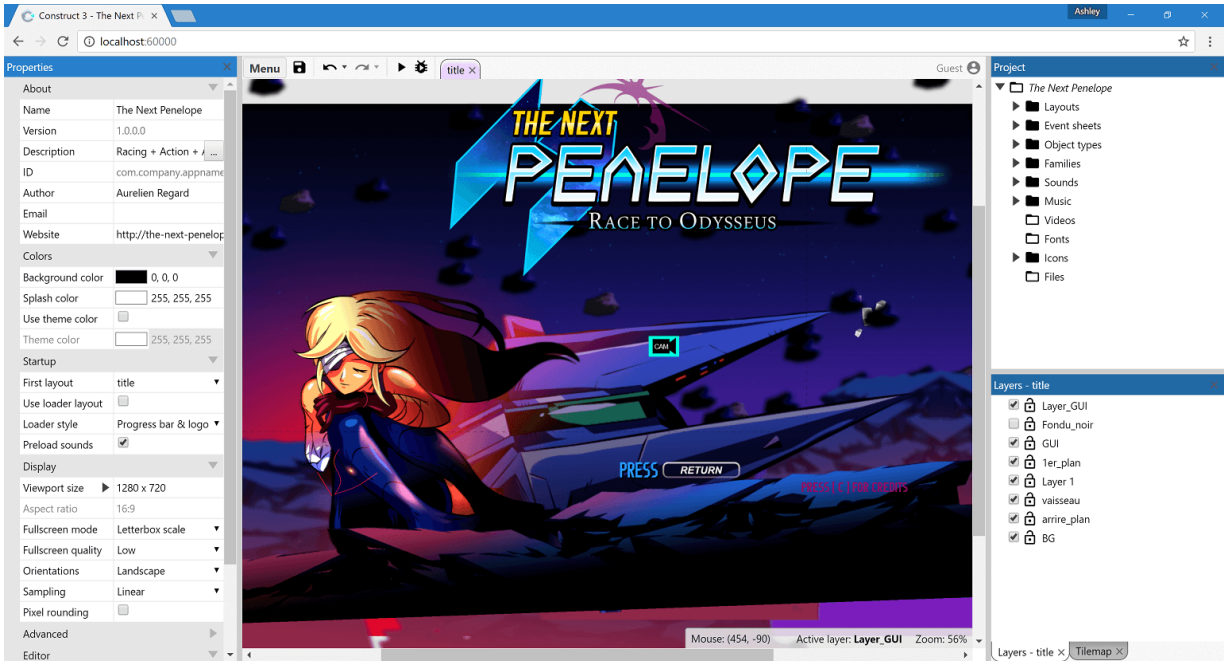


Figura 13.2: Construct 3 rodando dentro do Chrome

A principal vantagem é que não existe mais a necessidade de instalar nada em máquina, pois é tudo feito pela internet. Entretanto, não é possível trabalhar offline.

13.2 Novidades

As principais novidades são:

- **Execução do Construct 3 direto no navegador:** basta fazer login na conta e começar a construir os jogos.
- **Multiplataforma:** pode ser executado pelo Windows, Mac, Linux, Chrome OS e Android.
- **Multi-idiomas:** suporta a tradução completa para outras línguas.
- **Uso em celular:** usando o Android, é possível ajustar, visualizar e rever os projetos pelo seu celular, mas só se o dispositivo tiver o navegador Chrome instalado.

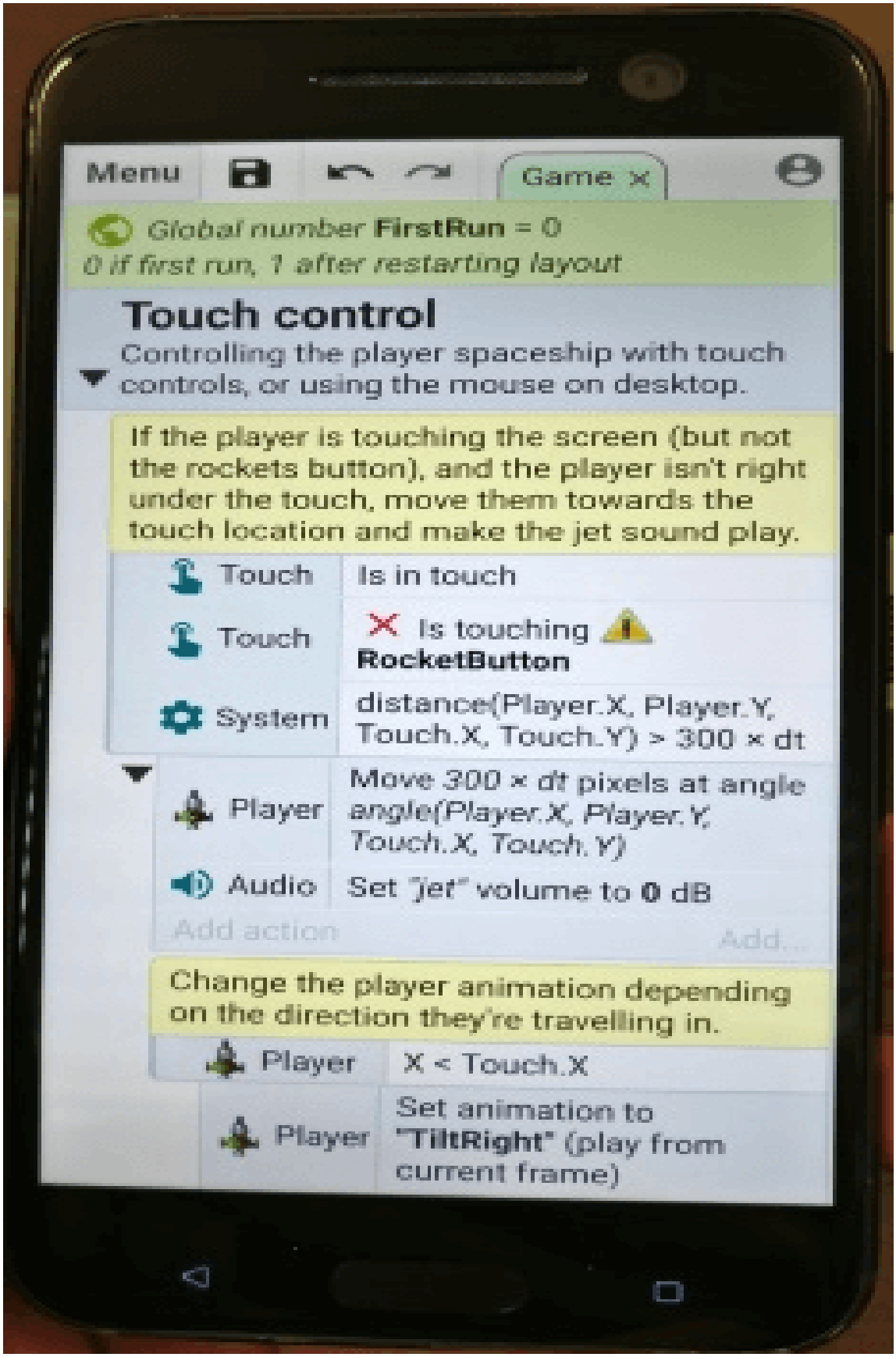

Menu
Game
Global number FirstRun = 0
0 if first run, 1 after restarting layout
Touch control
Controlling the player spaceship with touch controls, or using the mouse on desktop.
If the player is touching the screen (but not the rockets button), and the player isn't right under the touch, move them towards the touch location and make the jet sound play.
Touch
Is in touch
Touch
Is touching
RocketButton
System
distance(Player.X, Player.Y, Touch.X, Touch.Y) > 300 × dt
Player
Move 300 × dt pixels at angle angle(Player.X, Player.Y, Touch.X, Touch.Y)
Audio
Set "jet" volume to 0 dB
Add action
Add...
Change the player animation depending on the direction they're travelling in.
Player
X < Touch.X
Player
Set animation to "TiltRight" (play from current frame)

Figura 13.3: Construct 3 rodando no Android

- **Salvar na nuvem:** salve os projetos diretamente no Dropbox, Google Drive ou OneDrive, mantendo-os seguros.

- **Novos editores de dados:** configure dados para o seu jogo usando o novo editor de dicionário, editor de arrays e editor de texto.
- **Estatísticas do projeto:** veja as estatísticas com detalhes do seu jogo no novo diálogo dedicado, pela `Barra de projetos`.
- **Pré-visualização remota:** possibilidade de compartilhar o seu jogo por URL, permitindo também que ele seja testado em diversos dispositivos.
- **Pesquisa aprimorada:** recurso para pesquisar eventos por todo o projeto e também encontrar referências que permitem visualizar uma lista abrangente de todos os lugares em que um objeto, uma `Behavior` ou até uma variável é usada no jogo.
- **Design da folha de eventos atualizado:** foram incluídos novos botões para adicionar mais conteúdos, exibições de expressões mais claras, um modo de exibição compacto para celular e uma nova visualização de impressão.
- **Nova interface do editor de animações:** mais simples, com tudo em uma única janela. Nova paleta de cores, melhor suporte de arrastar e soltar, exibição de grade e mais opções de exportação.
- **Uso reduzido de memória:** arquitetura melhorada combinada com o carregamento sob demanda, de modo que apenas o conteúdo aparente é carregado na memória, ajudando a garantir que até projetos pesados sejam abertos em dispositivos sem muita memória.
- **Compatível com versões anteriores:** é possível importar os jogos do Construct 2 para o Construct 3.
- **Maior precisão de aceleração:** certifique-se de que seus movimentos, como o comportamento da plataforma, tenham uma melhor previsibilidade e precisão ao usar a precisão de

movimento aprimorada. Isso ajuda a garantir que as coisas como a altura do salto sejam mais consistentes.

Para obter mais informações, acesse o site oficial, em https://www.construct.net/br.